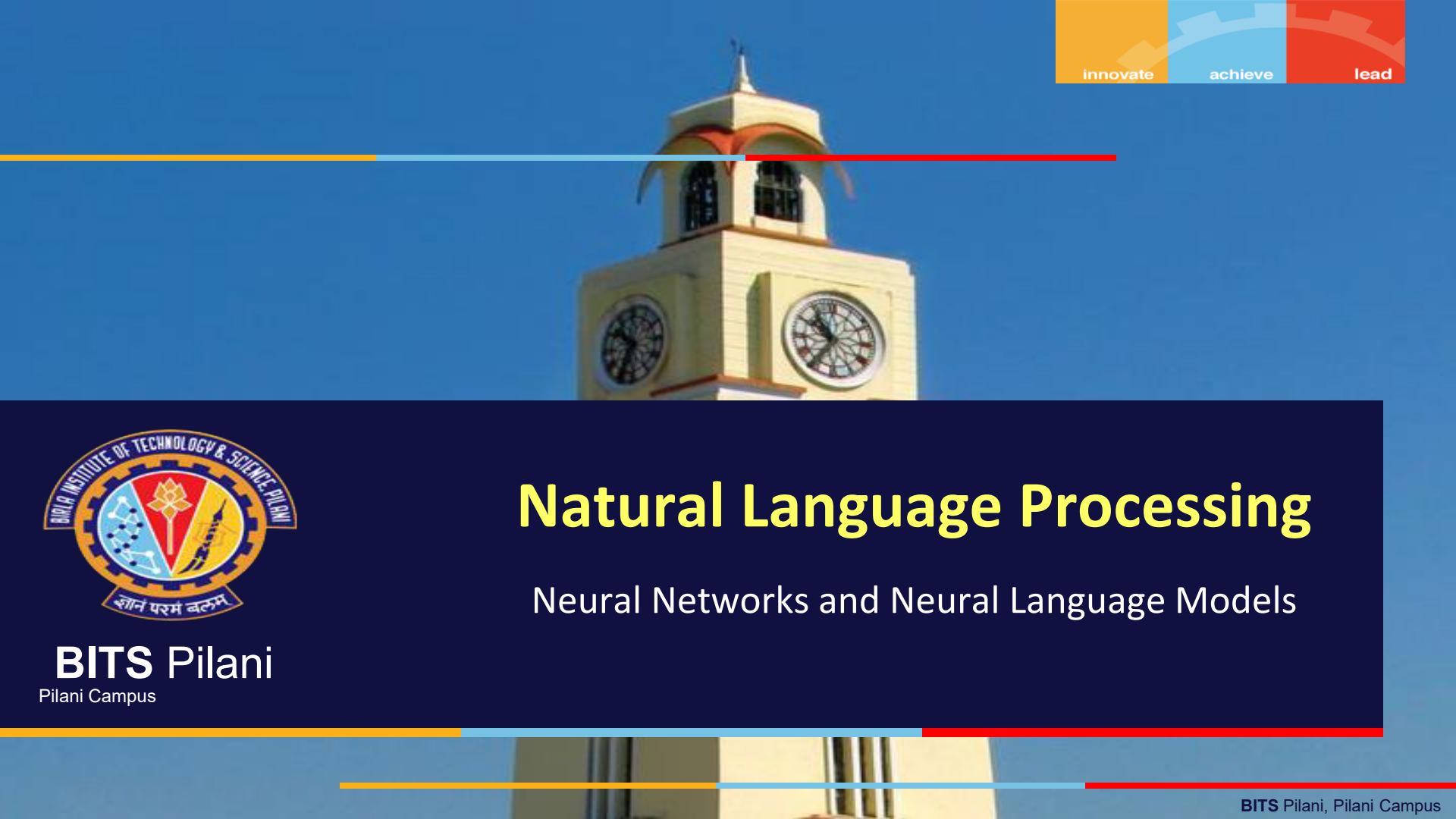


innovate achieve lead

BITS Pilani
Pilani Campus

Natural Language Processing

Neural Networks and Neural Language Models

innovate achieve lead

BITS Pilani

Pilani Campus

Grateful Acknowledgment: Source of this slide deck

Jurafsky and Martin: *https://web.stanford.edu/~jurafsky/slp3/*
https://web.stanford.edu/~jurafsky/slp3/slides/7_NN_Apr_28_2021.pptx

Session Content

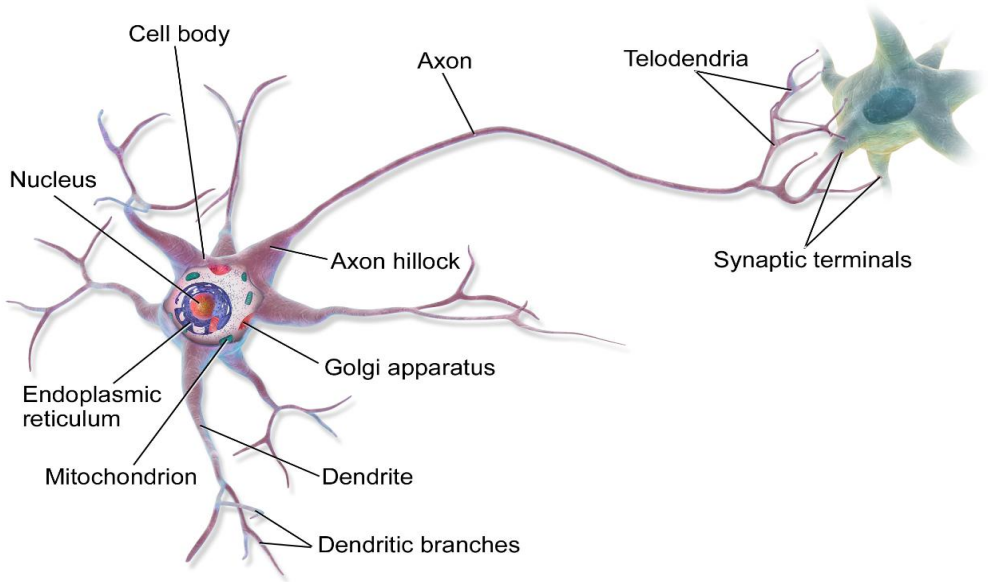
innovate achieve lead

Neural Networks and Neural Language Models

- Units in Neural Networks
- The XOR problem
- Feedforward Neural Networks
- Applying Feedforward Neural Networks to NLP tasks
- Training Neural Networks: Overview
- Computation Graphs and Backpropagation
- Neural Language Models

A Biological Neuron

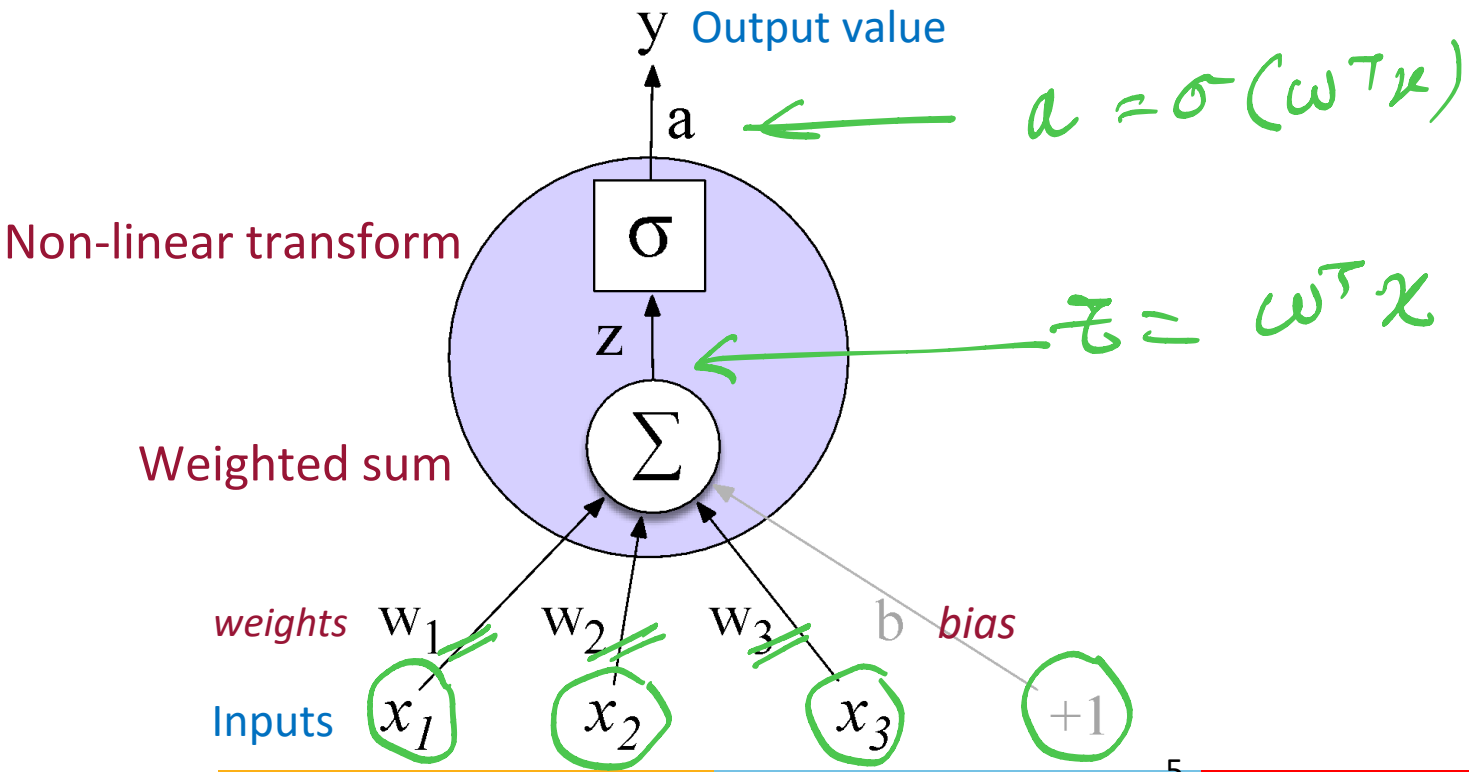
in the human brain



By BruceBlaus - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=28761830

Neural Network Unit

Inspired by the Neuron



Neural unit

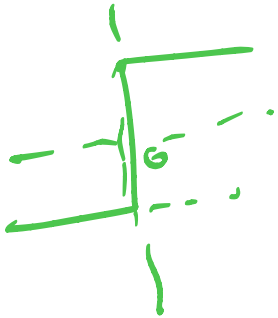
- Weighted sum of inputs, plus a bias:

$$z = b + \sum_i w_i x_i$$

$$z = W^T x + b$$

- Non-linear activation:

$$y = a = f(z)$$

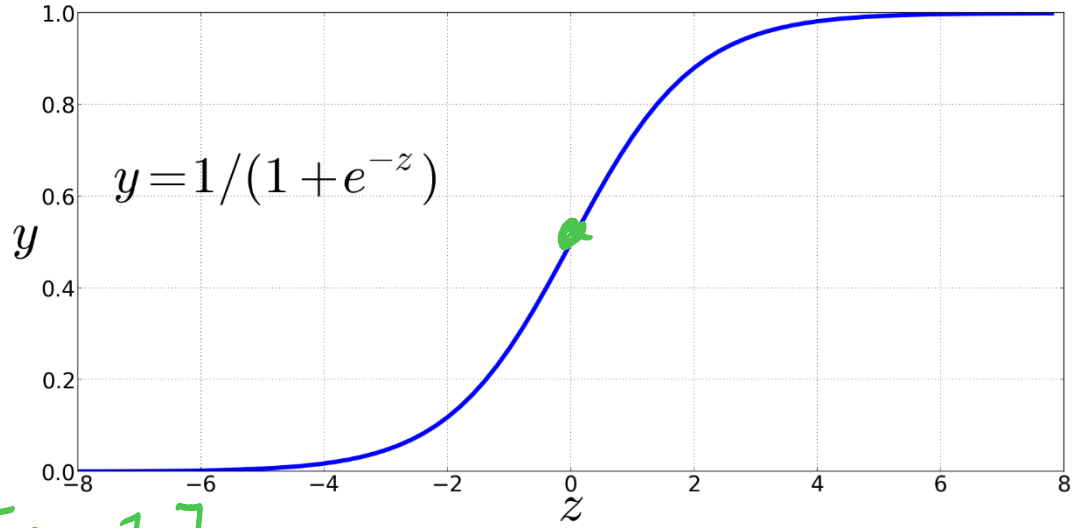


Activation Function - Sigmoid

We've already seen the sigmoid for logistic regression:

Sigmoid

$$y = s(z) = \frac{1}{1 + e^{-z}}$$

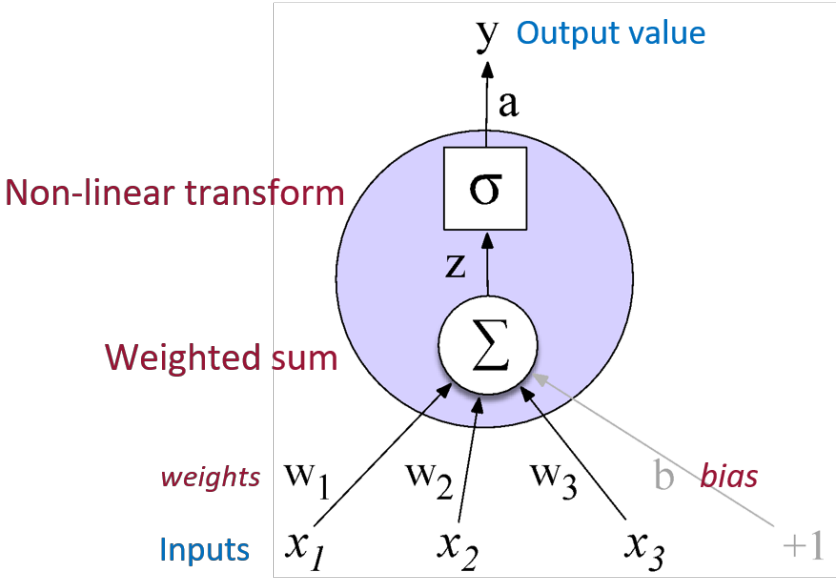


$[-\infty, \infty] \rightarrow [0, 1]$

Putting it together

$$y = s(w \cdot x + b) = \frac{1}{1 + \exp\left(-(w \cdot x + b)\right)}$$

$$y = \sigma\left([w_1 \ldots w_n] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + b \right)$$



An example

- *Suppose a unit has:*
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- What happens with input x:
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) =$$

An example

- *Suppose a unit has:*
  - $w = [0.2, 0.3, 0.9]$
  - $b = 0.5$
- What happens with the following input x?
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

An example

- *Suppose a unit has:*
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- What happens with input x:
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

$$\frac{1}{1 + e^{-(.5 \cdot .2 + .6 \cdot .3 + .1 \cdot .9 + .5)}} =$$

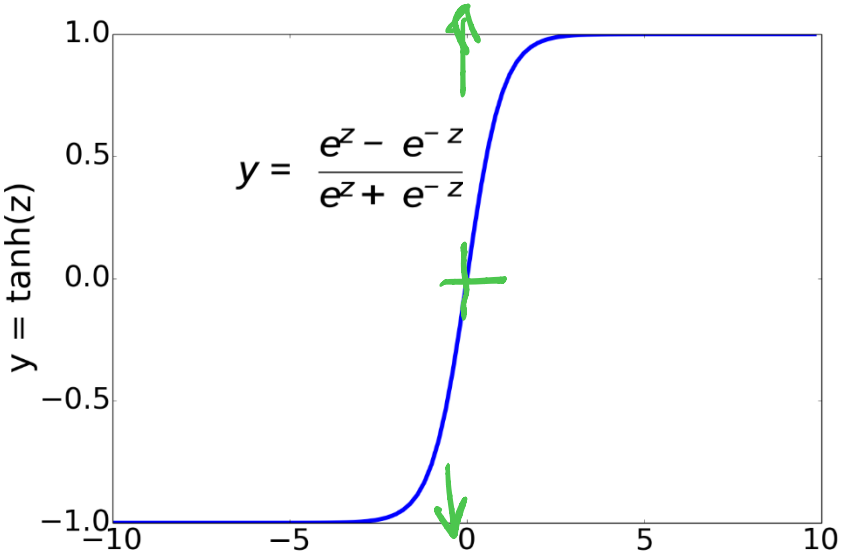
An example

- *Suppose a unit has:*
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- What happens with input x:
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

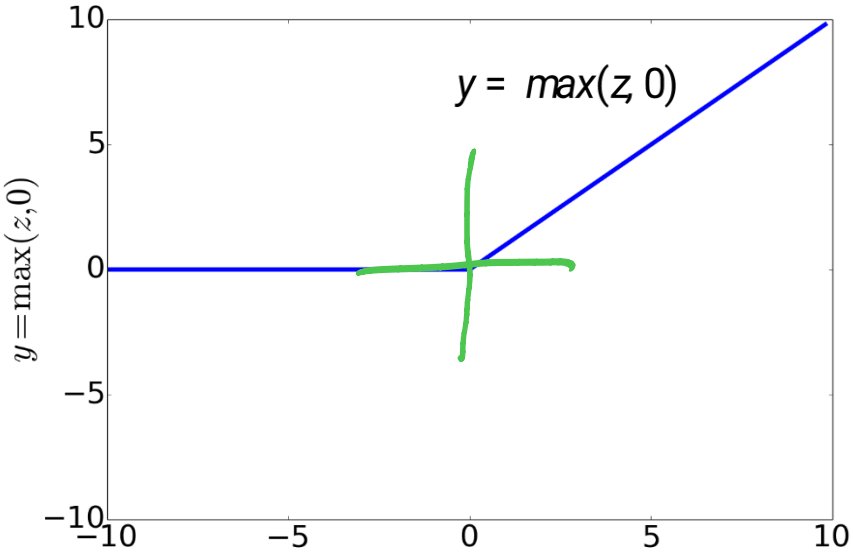
$$\frac{1}{1 + e^{-(.5 \cdot .2 + .6 \cdot .3 + .1 \cdot .9 + .5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

Non-Linear Activation Functions besides Sigmoid



tanh

Most Common:



ReLU
Rectified Linear Unit

The XOR Problem:

A Limitation of a Single Neural Unit

Minsky and Papert (1969)

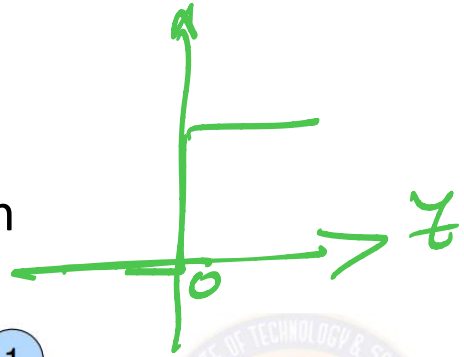
- Can a neural units compute every simple function?

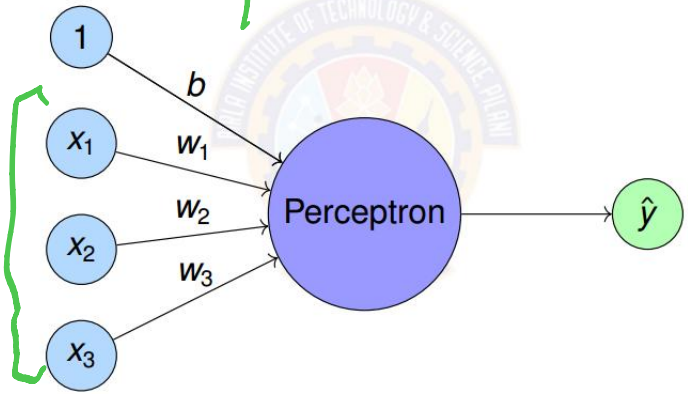
AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

Perceptron: A simple Neural Unit

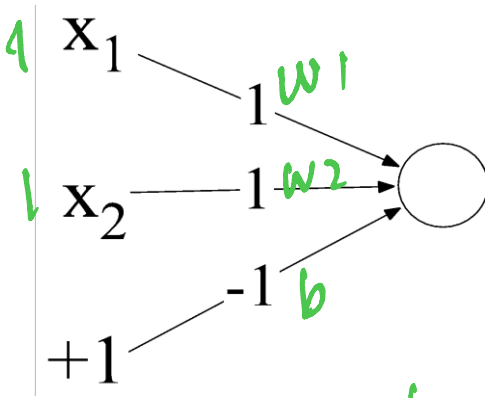
- A simple neural unit
  - Computes a weighted sum + bias
  - Uses a hard threshold (step) activation
  - Produces a binary output (0 or 1)

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



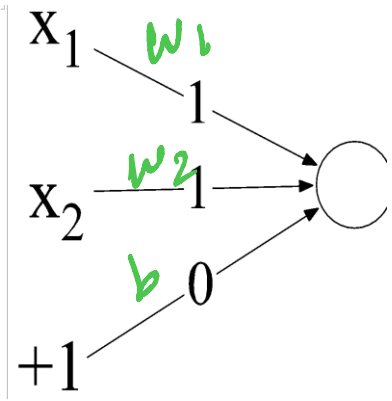


Perceptrons Can Represent AND and OR



AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



OR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

Can a Perceptron represent XOR?

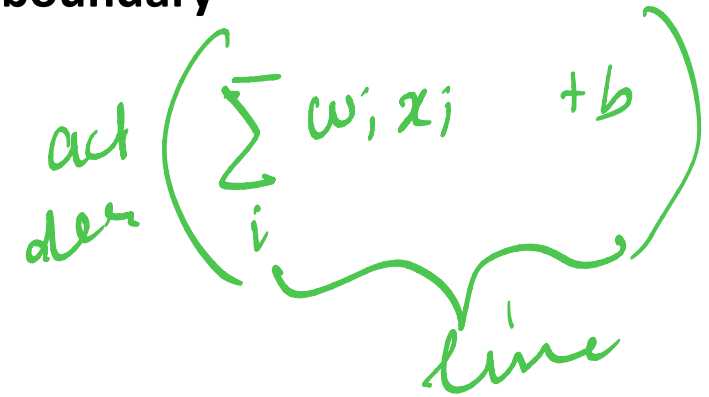
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

- Pause and try to find weights *w1, w2, b*
- No single perceptron can represent XOR.

Why? Perceptrons are linear classifiers

A perceptron defines a linear decision boundary

- $w_1 x_1 + w_2 x_2 + b = 0$

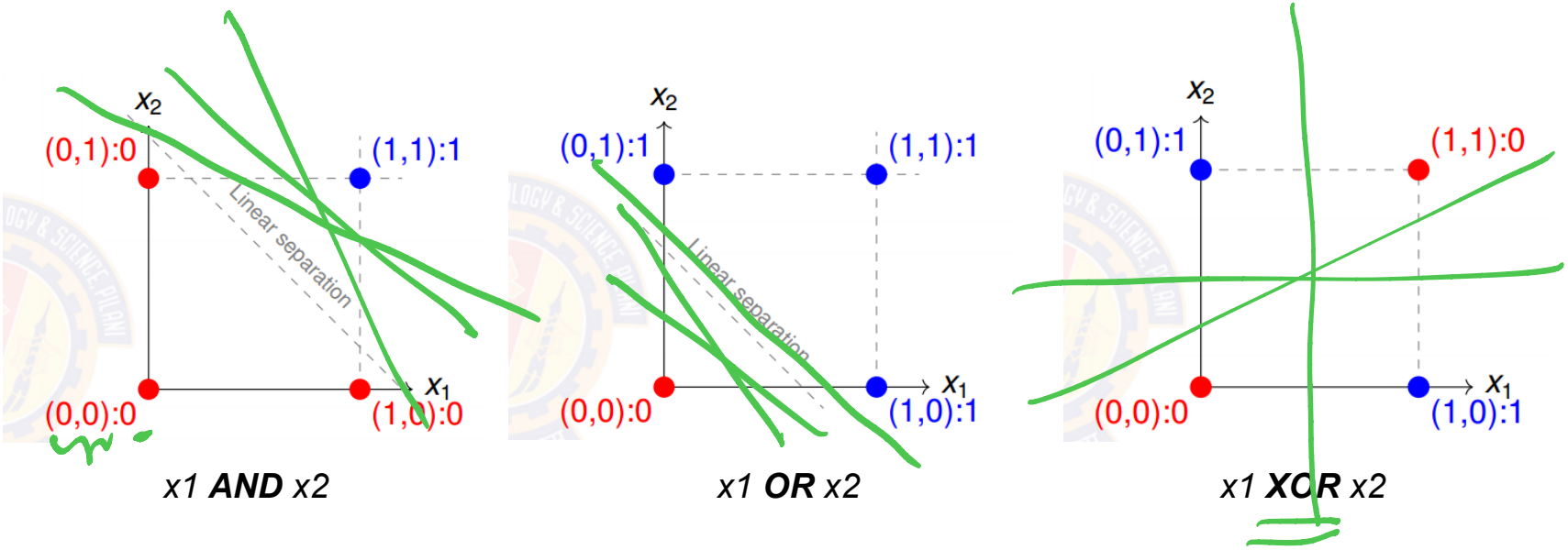


For two inputs, this is a line

This line acts as a decision boundary

- 0 if input maps to one side of the line
- 1 if it maps to other side of the line

Decision boundaries – AND, OR, XOR

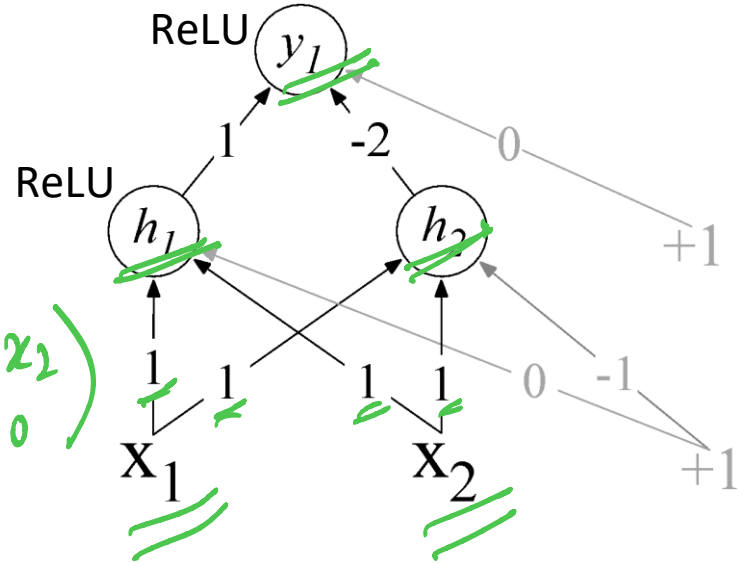


XOR is not a **linearly separable** function!

Solution to the XOR problem

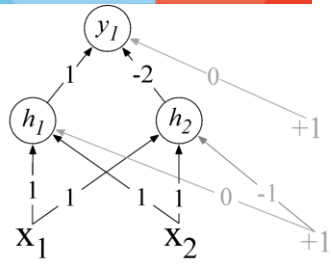
- XOR can't be calculated by a single perceptron
- XOR can be calculated by a layered network of units.

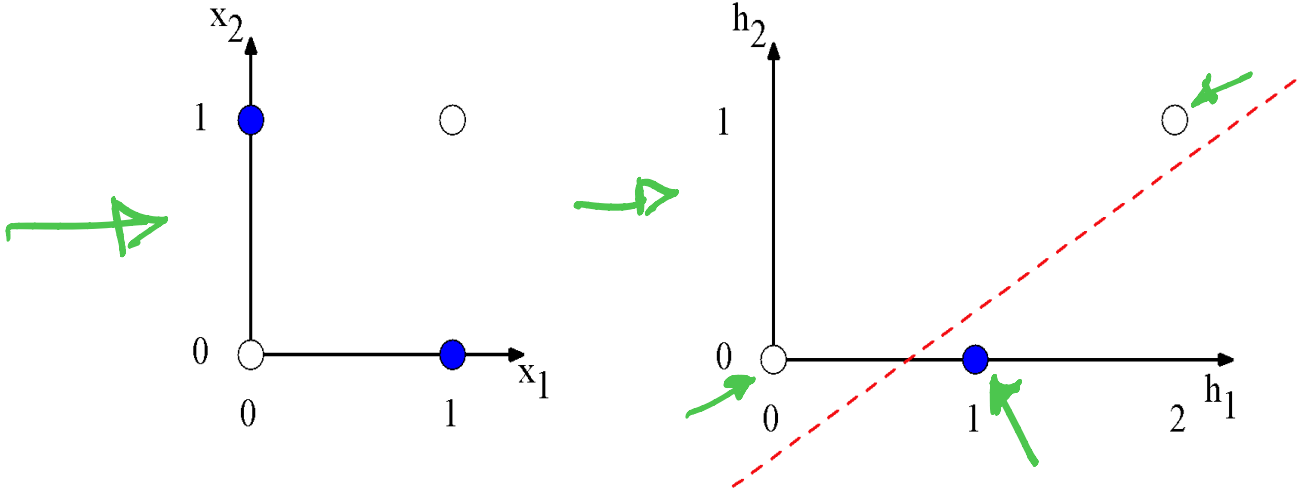
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



$h_1 = g(x_1 + x_2 + 0)$

The hidden representation *h*





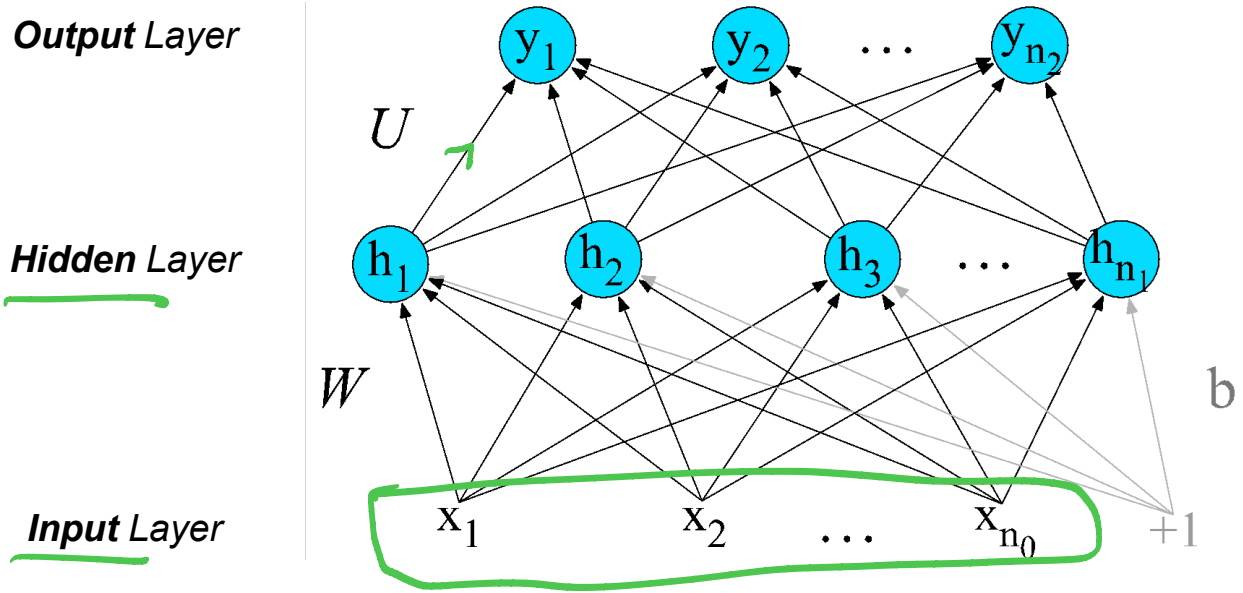
a) The original *x* space

b) The new (linearly separable) *h* space

(With learning: hidden layers will learn to form useful representations)

Feedforward Neural Networks

- Also called **multi-layer perceptrons** (or **MLPs**)

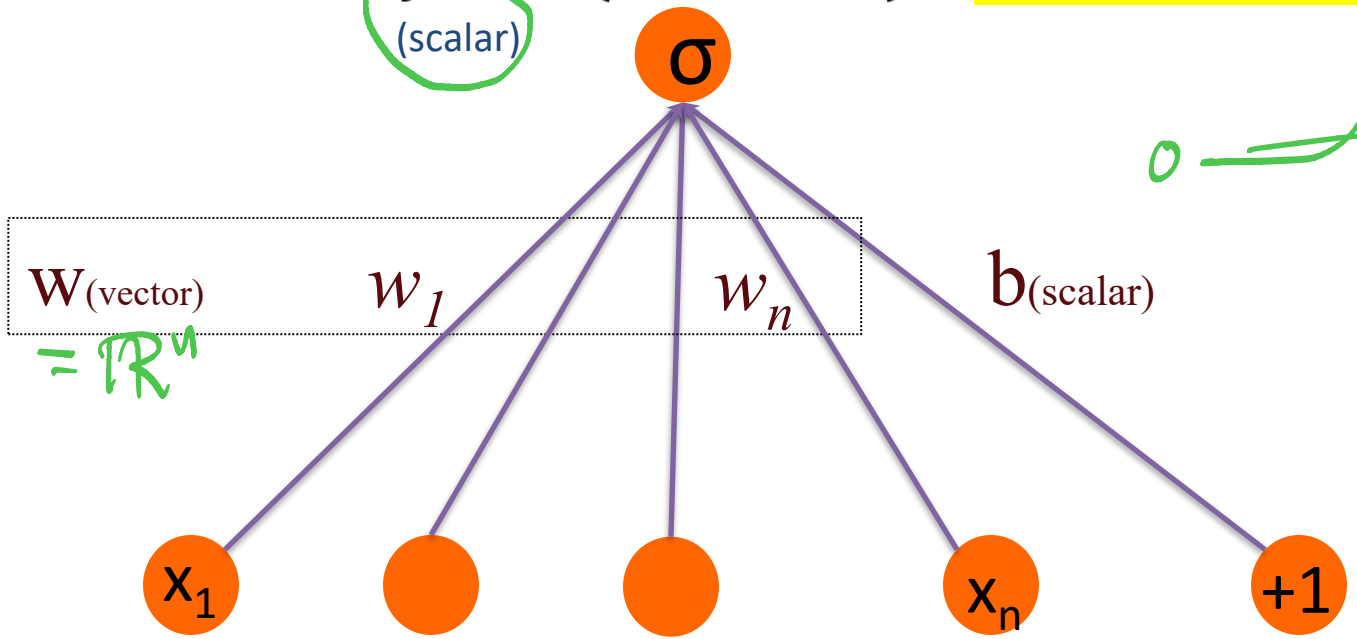


Binary Logistic Regression: A 1-layer Network

Output layer
(σ node)

$y = \sigma(w \cdot x + b)$
(scalar)

(we don't count the input layer while counting layers!)



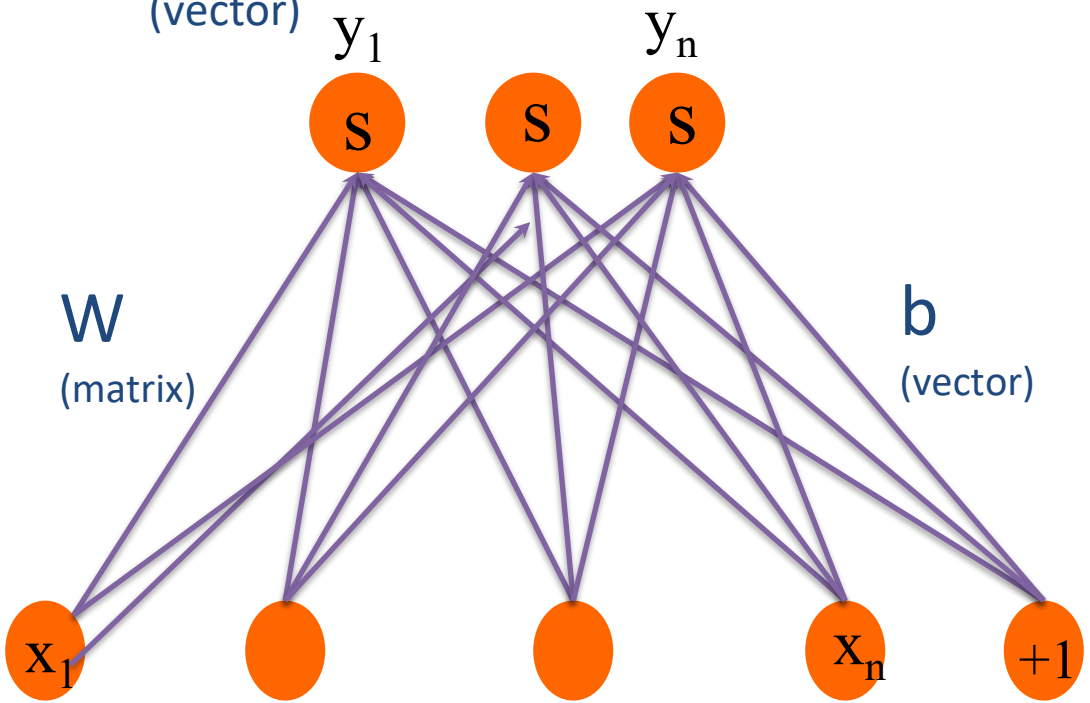
Input layer

1-Layer Network with Multiple Outputs

Output layer
(softmax nodes)

$$y = \text{softmax}(Wx + b)$$

(vector)



Input layer

softmax: a generalization of sigmoid

- For a vector *z* of dimensionality *k*, the softmax is:

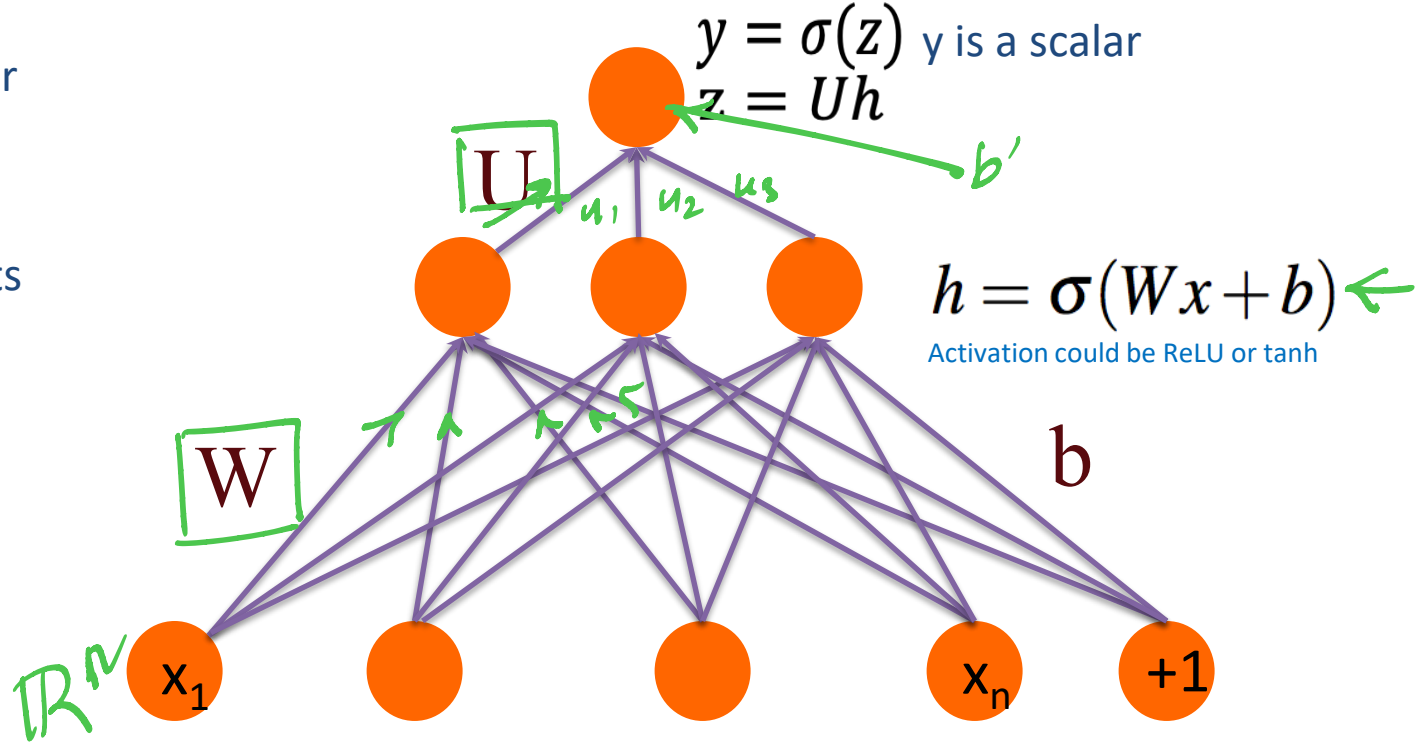
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \le i \le k$$

- Example:

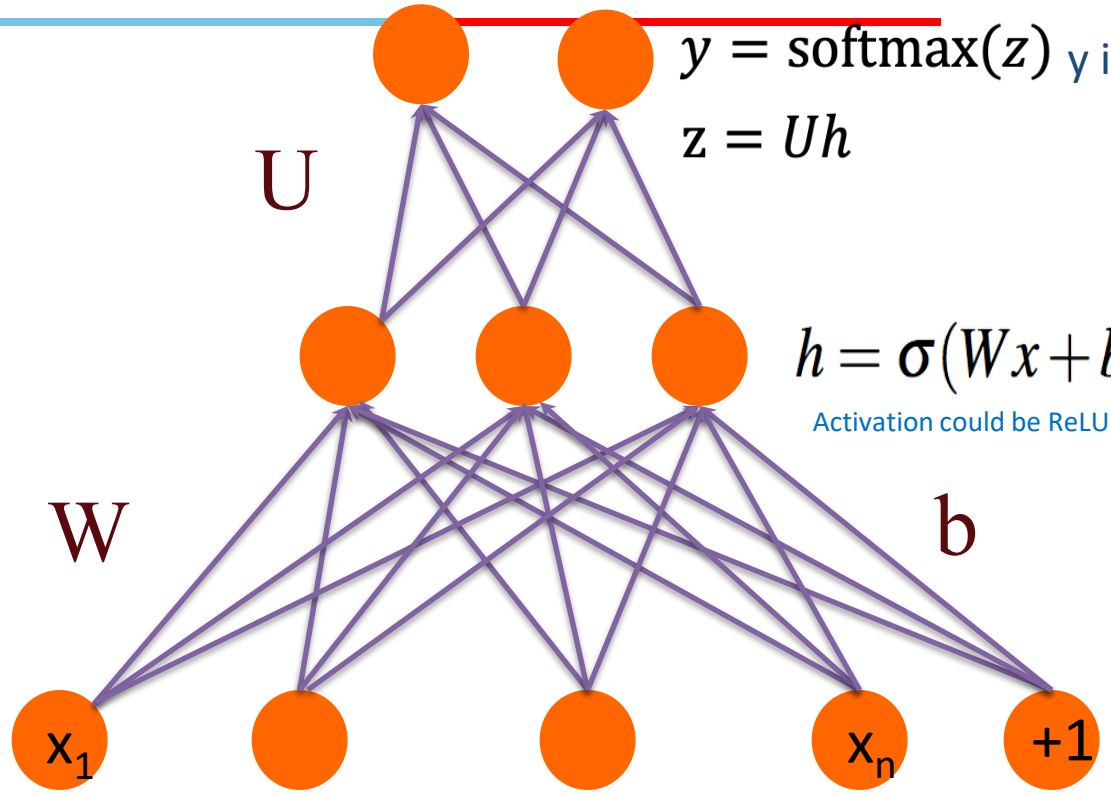
z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]

softmax(***z***) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]

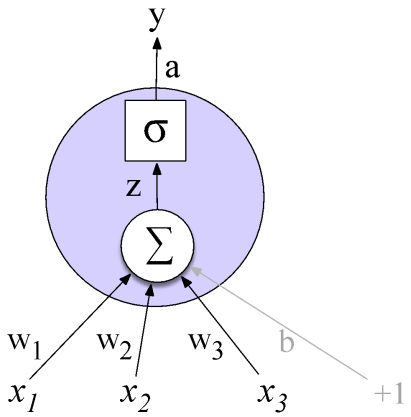
Two-Layer Network with scalar output



Two-Layer Network with softmax output



Multi Layer Notation



$$
\begin{aligned}
z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]} \\
a^{[1]} &= g^{[1]}(z^{[1]}) \\
z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\
a^{[2]} &= g^{[2]}(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

for i **in** 1..n

$$
\begin{aligned}
z^{[i]} &= W^{[i]} a^{[i-1]} + b^{[i]} \\
a^{[i]} &= g^{[i]}(z^{[i]}) \\
\hat{y} &= a^{[n]}
\end{aligned}
$$

logits

o → sigmoid

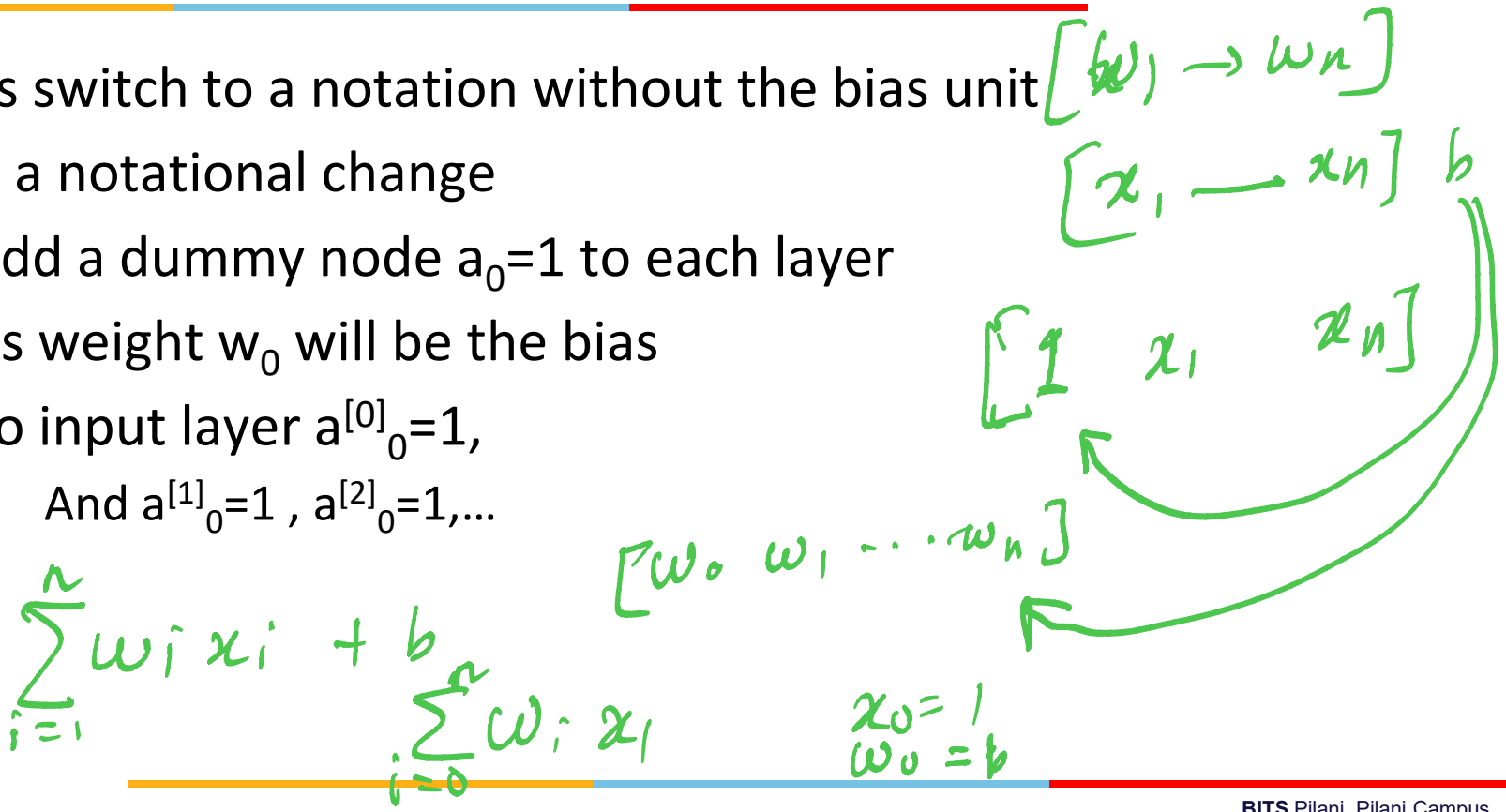
h → tanh, relu

tanh, relu

Replacing the bias unit

- Let's switch to a notation without the bias unit
- Just a notational change

1. Add a dummy node $a_0=1$ to each layer
2. Its weight w_0 will be the bias
3. So input layer $a^{[0]}_0=1$,
   - And $a^{[1]}_0=1$, $a^{[2]}_0=1$,…



Replacing the bias unit

$$x = x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx + b)$$

$$\sigma\left(\sum_{i=1}^{n_0} W_{ji}x_i + b_j\right)$$

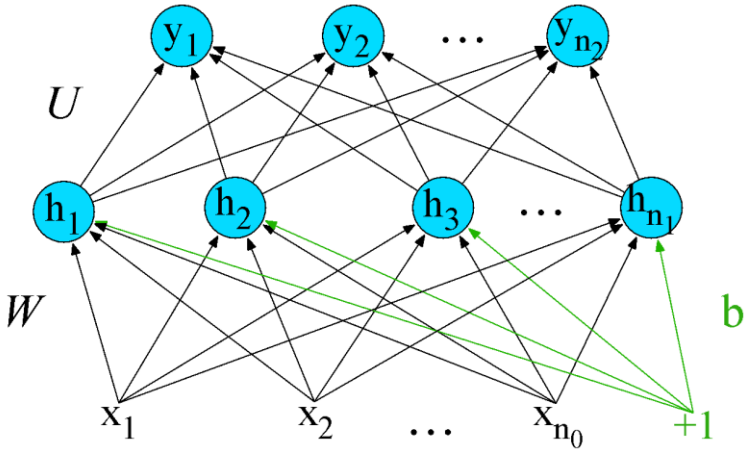
$$x = x_0, x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx)$$

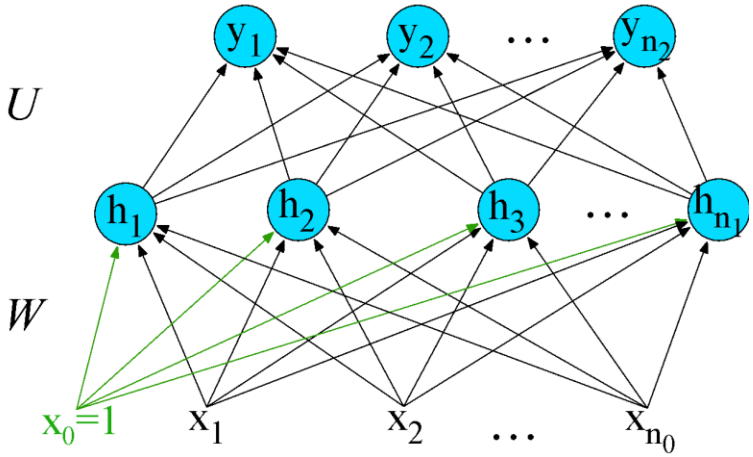
$$\sigma\left(\sum_{i=0}^{n_0} W_{ji}x_i\right)$$

Replacing the bias unit

Instead of:



We'll do this:



Feedforward Networks for NLP

- Let's consider two simplified applications:

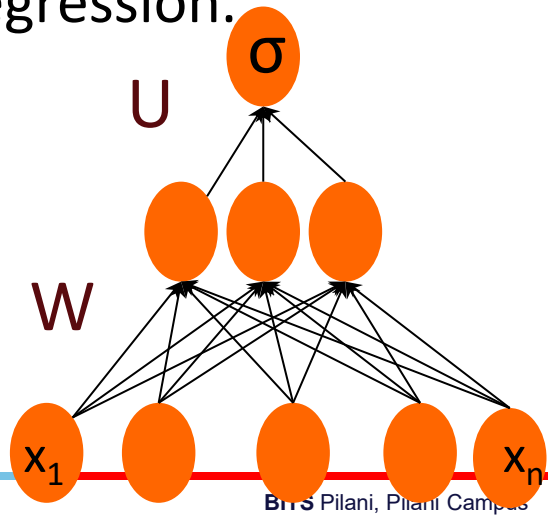
  1. Text classification (sentiment analysis) Positive Negative
  2. Language modeling

Example 1: Sentiment Classification

- **Input:** features extracted from the document
  Output: positive / negative

- A single-layer network is essentially logistic regression.

- Input layer are binary features as before
- Output layer is 0 or 1

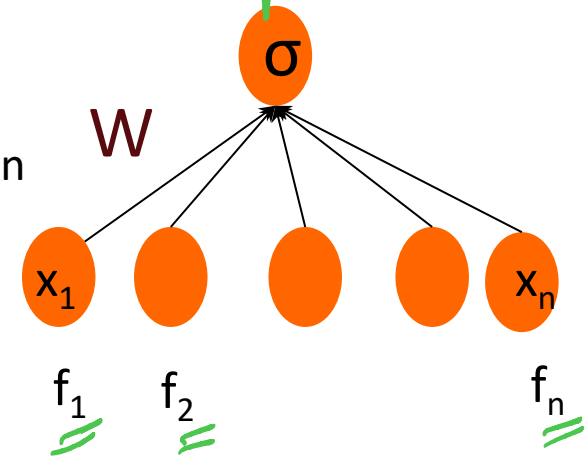


Sentiment Features

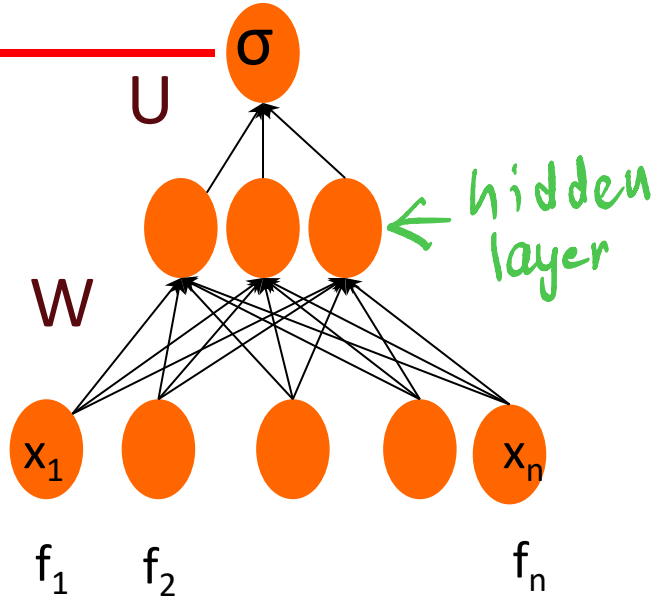
Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Adding a Hidden Layer: Nonlinear Feature Interactions

Logistic Regression



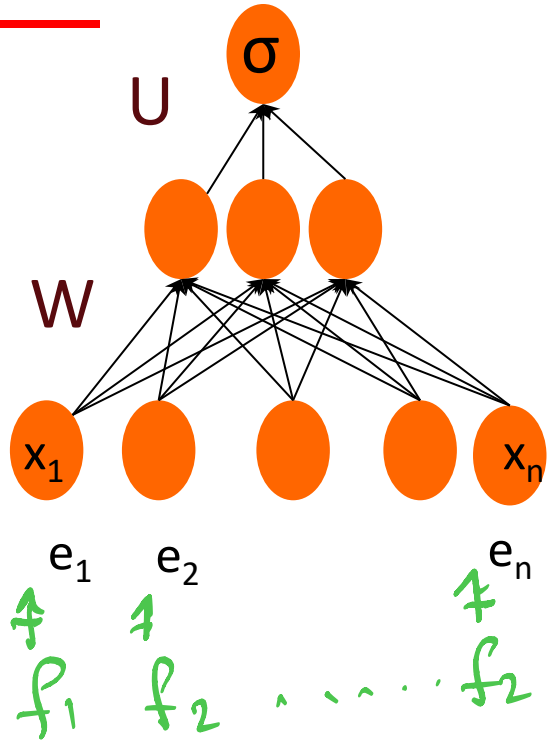
2-layer Feedforward Network



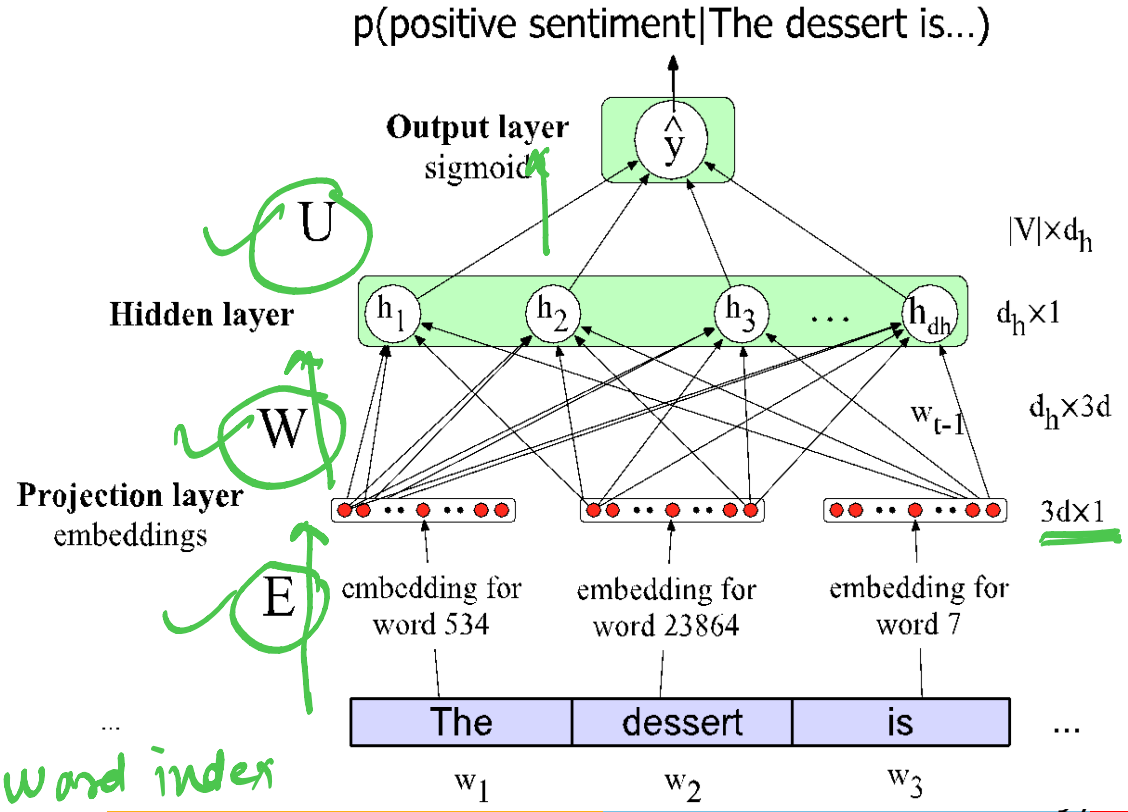
- Logistic regression uses a linear decision boundary
- A hidden layer allows nonlinear interactions between features
- This may improve classification performance

Even better: representation learning

- The real power of deep learning comes from the ability to **learn** features from the data.
- Instead of using hand-built human-engineered features for classification, learn useful representations directly from data.
- Word embeddings are one example.



Neural Net Classification with embeddings as input features



Challenge: Variable-length text

- This approach assumes text has **fixed length** (here, 3)
  - This is unrealistic
- Some simple solutions (*more sophisticated solutions later*)

1. Pad/Truncate to a fixed length (*e.g. max length review*)
   - If shorter, pad with zero embeddings
   - Truncate - if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings, or
   - Take the element-wise max of all the word embeddings
     - For each dimension, pick the max value from all words

Example 2: Neural Language Models (LMs)

Goal: Predict the next word given the preceding context.

$$P(w_t \mid w_1, \ldots, w_{t-1})$$

- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models

Modern Language Models use architectures such as Transformers, but we will first study a **simple feedforward neural LM**.

Feedforward Neural Language Model: Fixed Context

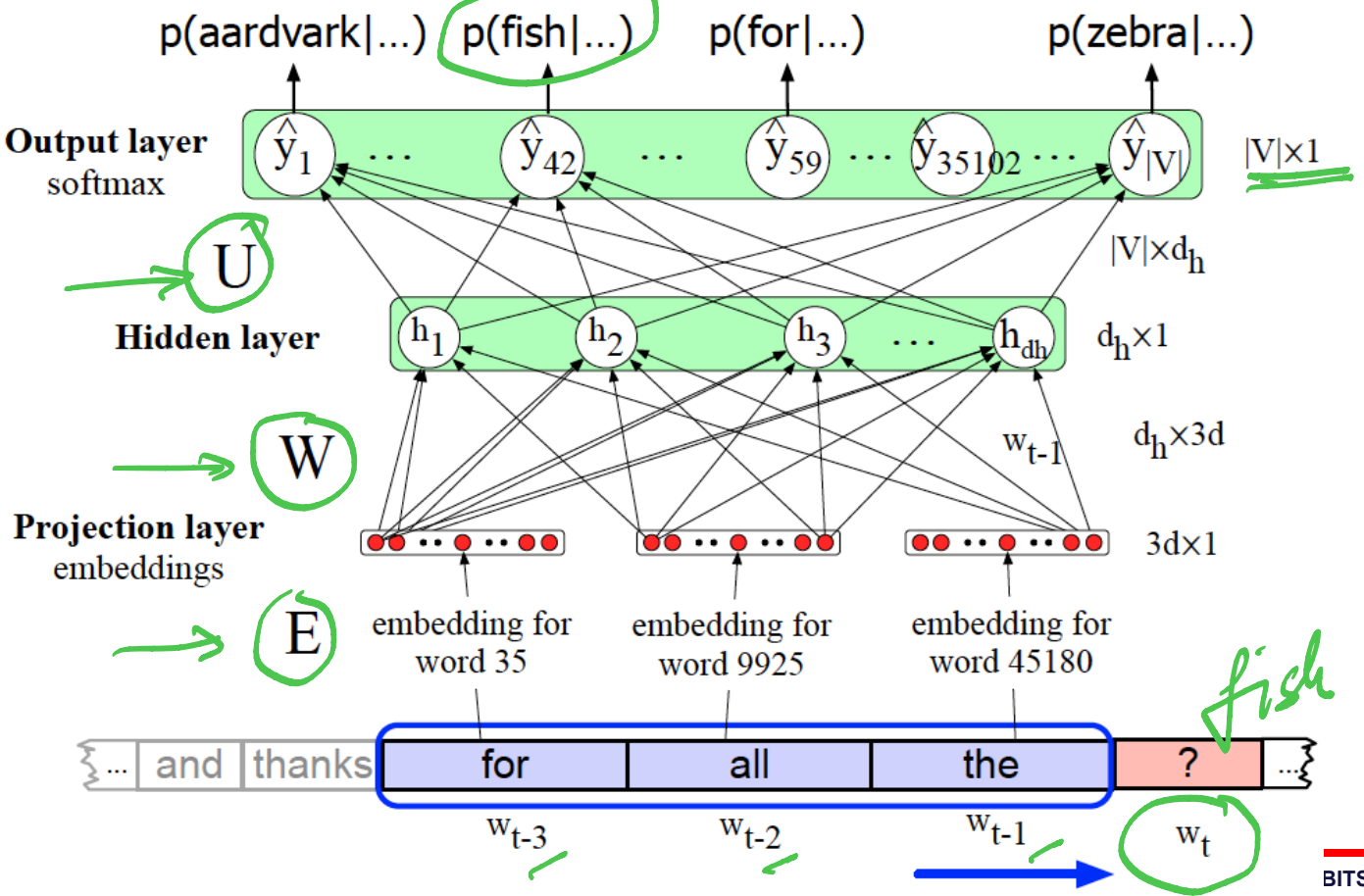
We need to **predict next word given a past sequence**, but we are dealing with sequences of arbitrary length.

- **Solution**: Sliding window, of fixed length

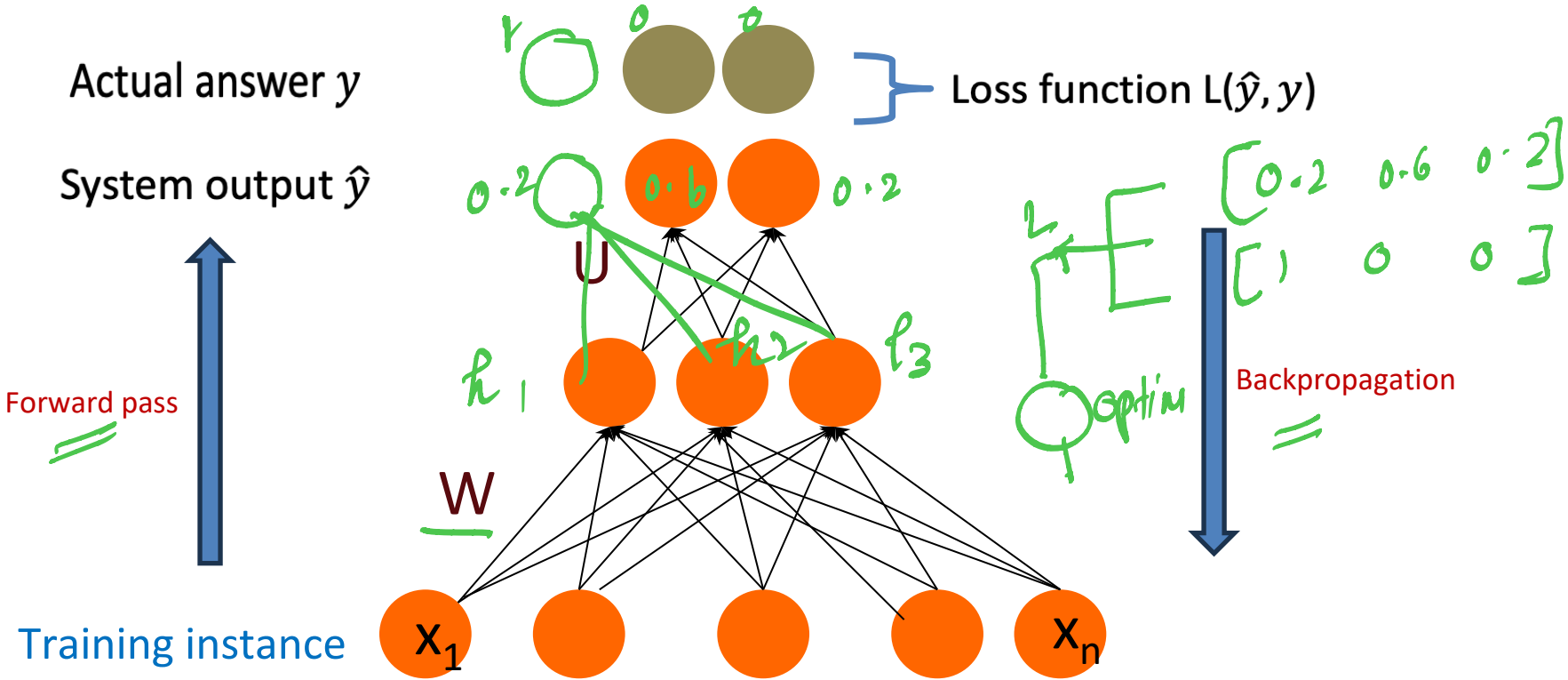
$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

- Like an n-gram LM, the feedforward NLM uses a fixed-size context window.
- But, words are represented by learned continuous vectors rather than discrete n-gram identities.

Neural Language Model



Intuition: training a 2-layer Network



Intuition: training a 2-layer network

- For every training tuple (x, y)
  - Run *forward* computation to find our estimate $\hat{y}$
  - Run *backward* computation to update weights:
    - For every output node
    - Compute loss L between true y and the estimated $\hat{y}$
    - For every weight w from hidden layer to the output layer
      - » Update the weight
    - For every hidden node
    - Assess how much blame it deserves for the current answer
    - For every weight w from input layer to the hidden layer
      - » Update the weight

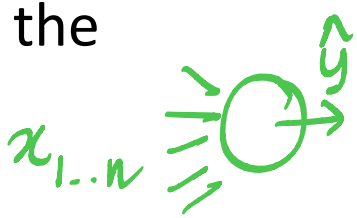
- $x \rightarrow \hat{y}$
- $L = f(y, \hat{y})$
- $\nabla_w L$
- $w' \leftarrow w + -\eta \nabla_w L$

Reminder: Loss Function for binary logistic regression

- A measure for how far off the current answer is to the right answer
- Cross entropy loss for logistic regression:

$$L_{\text{CE}}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

$$= -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$



$y=1$

$\hat{y} = p = 0.9$ ↑

$\hat{y} = 0.6$ ↑

- For an output distribution over |V| classes:

$$L = -\sum_{i=1}^{V} y_i \log \hat{y}_i$$

Reminder: gradient descent for weight updates

- Use the derivative of the loss function with respect to weights

$$\frac{d}{dw} L(f(x;w),y)$$

- To tell us how to adjust weights for each training item
  - Move them in the opposite direction of the gradient

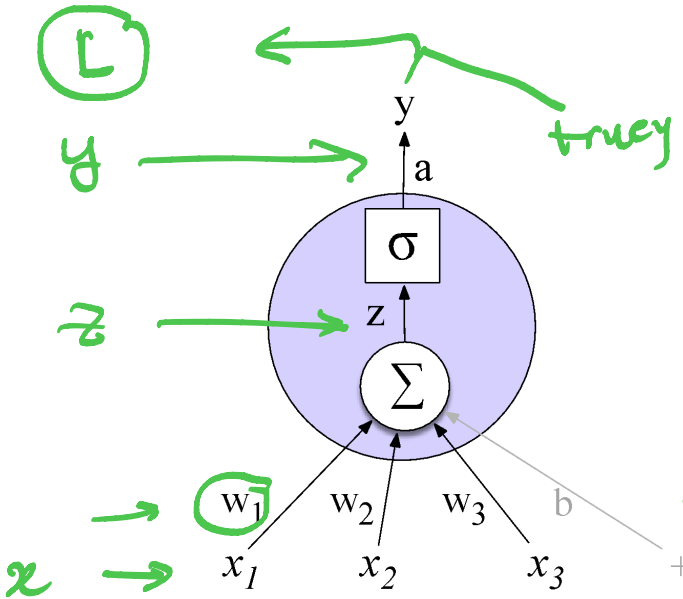
$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x;w),y)$$

  - For logistic regression

$$\frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

Backpropagation uses the Chain Rule

- For $f(x) = u(v(x))$, $\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$



$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}$$

How can I find that gradient for every weight in the network?

- The derivatives on the prior slide only give the updates for one weight layer: the last one.

- What about **deeper networks**?
- Lots of layers, different activation functions

- Solution: backpropagation through computation graphs
- Even more use of the chain rule.
- Computation graphs and backward differentiation.

Summary

- For training, we need the derivative of the loss with respect to weights in early layers of the network.
  - But loss is computed only at the very end of the network.

- Solution: **backward differentiation**
  - Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.

N-Gram LMs to Neural LMs

$$P(w_1, \ldots, w_T) = \prod_t P(w_t | w_1, \ldots, w_{t-1})$$

$$P(w_t | w_1, \ldots, w_{t-1}) \approx P(w_t | w_{t-N+1}, \ldots, w_{t-1})$$

$P(w_1 w_2 w_3 w_4)$

$= P(w_1) P(w_2|w_1)$

$P(w_3|w_2 w_1) \ldots$

The neural approach replaces discrete word/context representations with learned continuous representations.

Neural Probabilistic Language Models

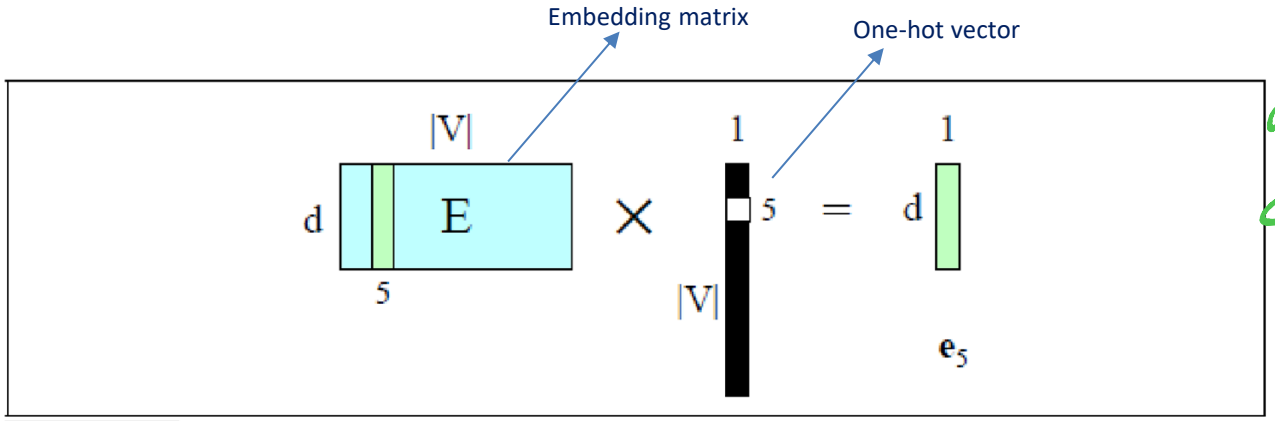
- Instead of treating words as tokens, exploit semantic similarity
  - Learn a distributed representation of words that will allow sentences like these to be seen as similar
    - The cat is walking in the bedroom. ← training
    - A dog was walking in the room. ← test
    - The cat is running in a room.
    - The dog was running in the bedroom.
    - etc.
- Use a neural net to represent the conditional probability function
  - $P(w_t|w_{t-n}, w_{t-n+1}, \dots, w_{t-1})$
- Learn the word representation and the probability function simultaneously

Word Embeddings

- Represent each of the N previous words as a one-hot vector of length |V|

$$x_i \in \mathbb{R}^{|V|} \longrightarrow e_i = Ex_i \longrightarrow e_i \in \mathbb{R}^d$$

dog [0 0 0 0 1]
cat [0 1 0 0 0]

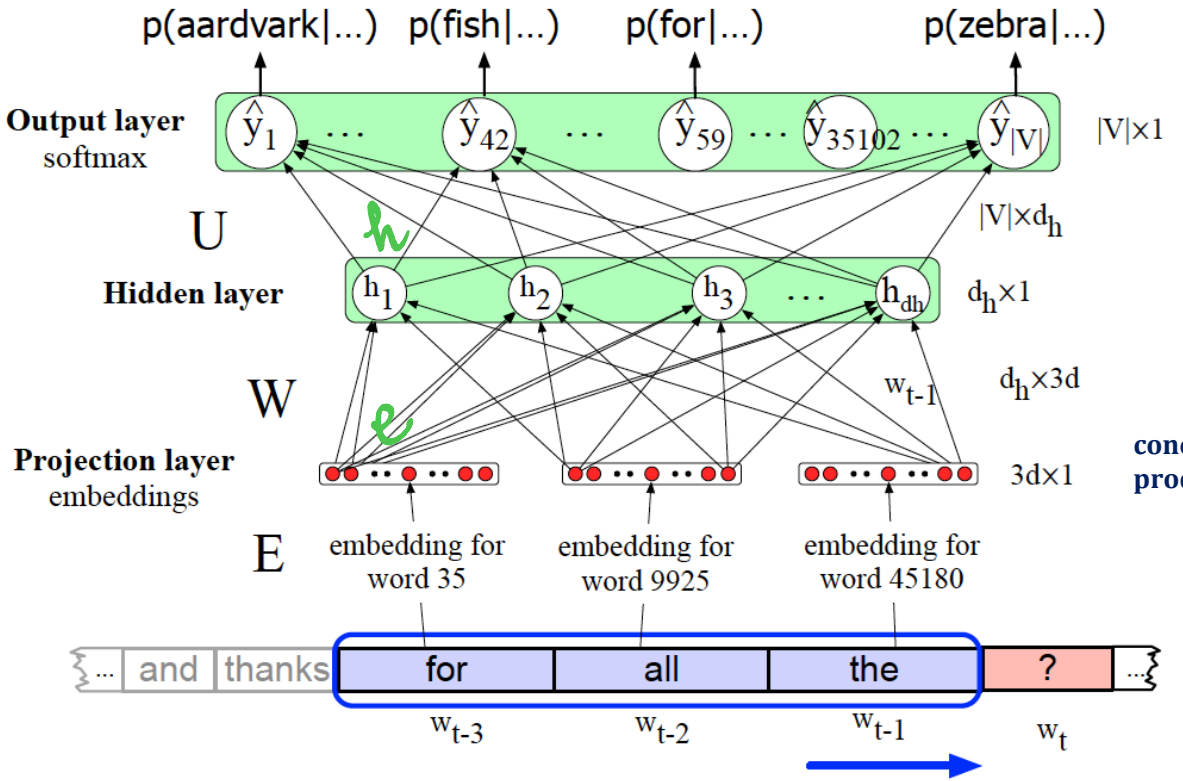


dog [0.6 0.1]
cat [0.7 0.3]

Why Neural LMs Generalize

- **N-gram LMs** treat words as discrete symbols
  - A new word/context combination may have little or no observed evidence.
- **Neural LMs** represent words using **learned embeddings.**
  - Similar words tend to have similar representations.
  - This allows the model to **generalize across similar contexts**.
- **Example**
  - **Training:**
    - I have to make sure that the **cat gets fed**.
  - **Test:**
    - I forgot to make sure that the **dog gets** ___.
  - An N-gram model may have no evidence for this unseen context.
  - A neural LM can exploit the similarity between the **cat** and **dog** embeddings and predict **"fed"**.

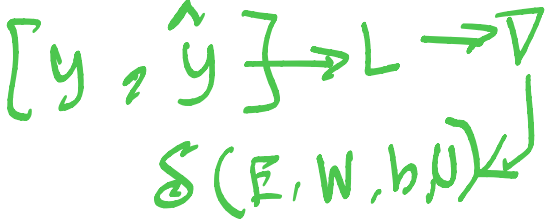
Neural Language Model



Equations:

$$e = [Ex_{t-3}; Ex_{t-2}; Ex_{t-1}]$$
$$h = \sigma(We + b)$$
$$z = Uh$$
$$\hat{y} = \text{softmax}(z)$$

concatenate 3 embeddings for the 3 context words to produce the embedding layer e



Training the Neural Language Model

What Parameters do we train?

- **Option 1 — Fixed embeddings** → training W, b, U
  - Initialize E from a pretrained embedding model (e.g. word2vec)
  - Keep E fixed
  - Train the remaining network parameters
- **Option 2 — Train embeddings jointly** → training W, b, U, E
  - Initialize E ← random weights
  - Update E along with the network
  - Usually preferred when enough task-specific data is available

N-gram vs NLM

	N-gram LM	Neural LM
Representation	Discrete	Distributed embeddings
Context	Fixed	Fixed for feedforward NLM
Generalization	Limited	Better across similar words
Training/inference	Simple	More computationally expensive
Interpretability	Higher	Lower

innovate achieve lead

BITS Pilani

Pilani Campus

From Neural Language Models to LLMs

From Neural Language Models to LLMs

- A neural language model learns to predict the next word:

$$P(w_t \mid w_1, \ldots, w_{t-1})$$

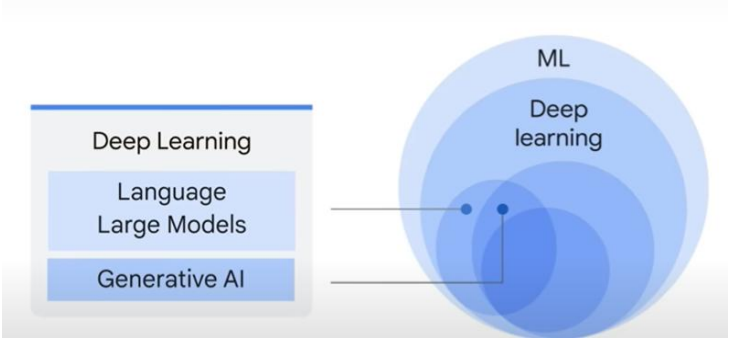
- What happens when we **scale this idea**, with:
  - Much larger training datasets
  - Much larger models
  - General-purpose language modeling
  - Pretraining followed by adaptation to specific tasks

pre-train

- Large Language Models (LLMs) are large-scale neural language models trained to acquire general-purpose language capabilities.

Large Language Models

- **Large**
  - Large training datasets
  - Large number of parameters
- **General-purpose**
  - Trained on broad language data
  - Intended to support many downstream tasks
- **Pretrained**
  - Learn general language patterns during pretraining
  - Can subsequently be adapted to specific tasks



Large Language Models - Examples

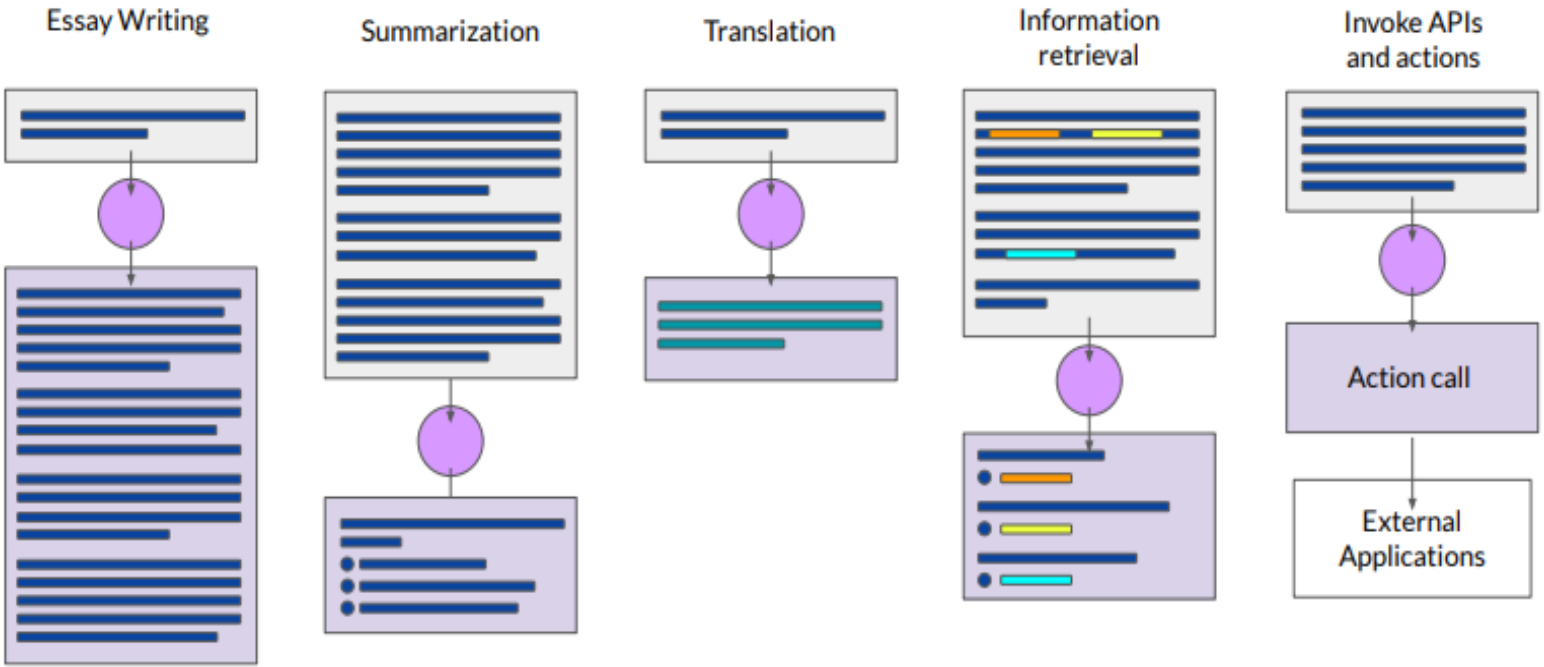
- **Proprietary / hosted**
  - **GPT** — OpenAI
  - **Claude** — Anthropic
  - **Gemini** — Google
- **Open-weight / self-hostable**
  - **Llama** — Meta
  - **Qwen** — Alibaba
  - **DeepSeek** — DeepSeek
- Model families differ in architecture, scale, training data, capabilities and deployment options

Why Use Large Pretrained Models?

- One pretrained model can support many tasks
- Task-specific adaptation may require relatively little labelled data
- Knowledge learned from large-scale data can be reused
- Increasing model/data scale has historically led to improved capabilities

Traditional ML	Pretrained LLM
Train a model for each task	Start from one general-purpose model
Task-specific training data	Adapt with relatively little task-specific data
Task-specific model	Reusable foundation

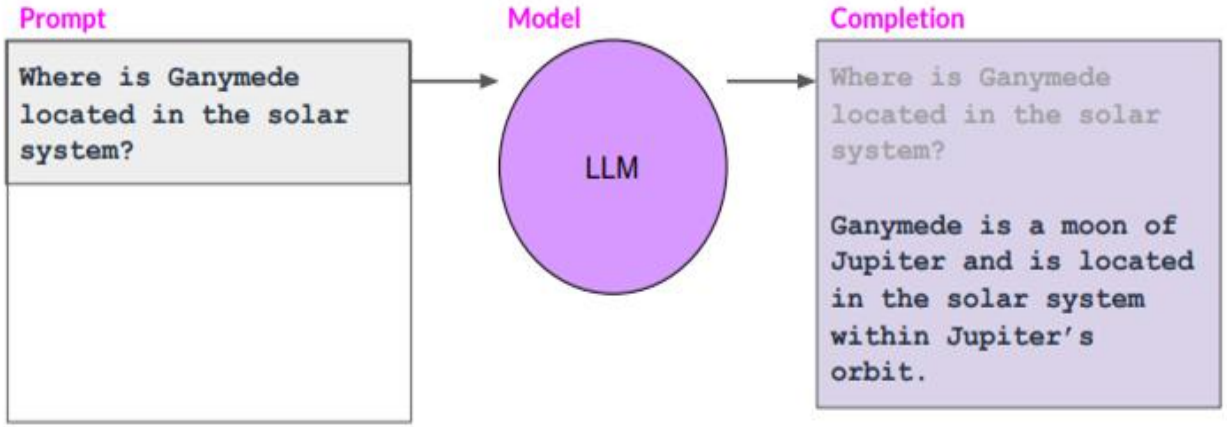
Large Languate Models – Use Cases



LLMs: Prompts -> Completion

The model generates a continuation based on the prompt and its learned language patterns.

The prompt provides the **context available** to the model.

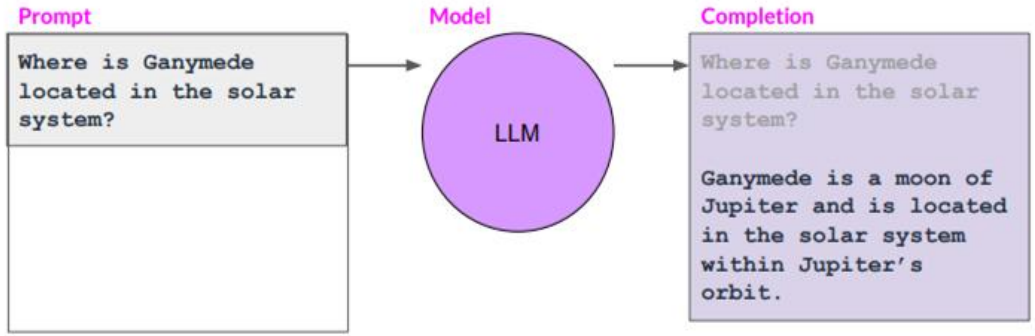

Prompt
Where is Ganymede located in the solar system?
Model
LLM
Completion
Where is Ganymede located in the solar system?
Ganymede is a moon of Jupiter and is located in the solar system within Jupiter's orbit.

LLMs: Prompting and Prompt Engineering

A prompt can specify:

- **Task** — what should the model do?
- **Context** — what information should it use?
- **Constraints** — how should it respond?
- **Examples** — what does a desired response look like?

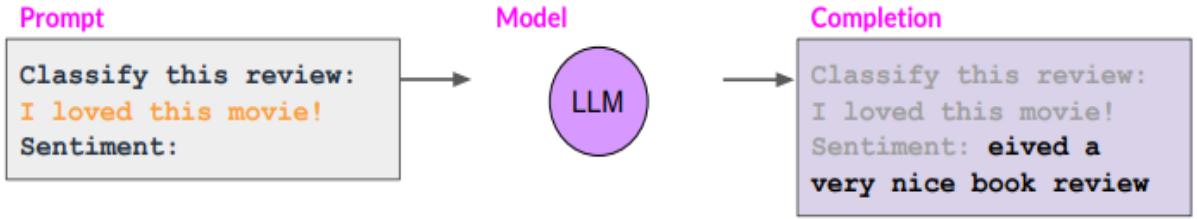
Prompt engineering is the design of prompts to elicit useful model behaviour**.**


Prompt
Where is Ganymede located in the solar system?
Model
LLM
Completion
Where is Ganymede located in the solar system?
Ganymede is a moon of Jupiter and is located in the solar system within Jupiter's orbit.

LLMs: In-Context Learning

Zero-shot Prompting

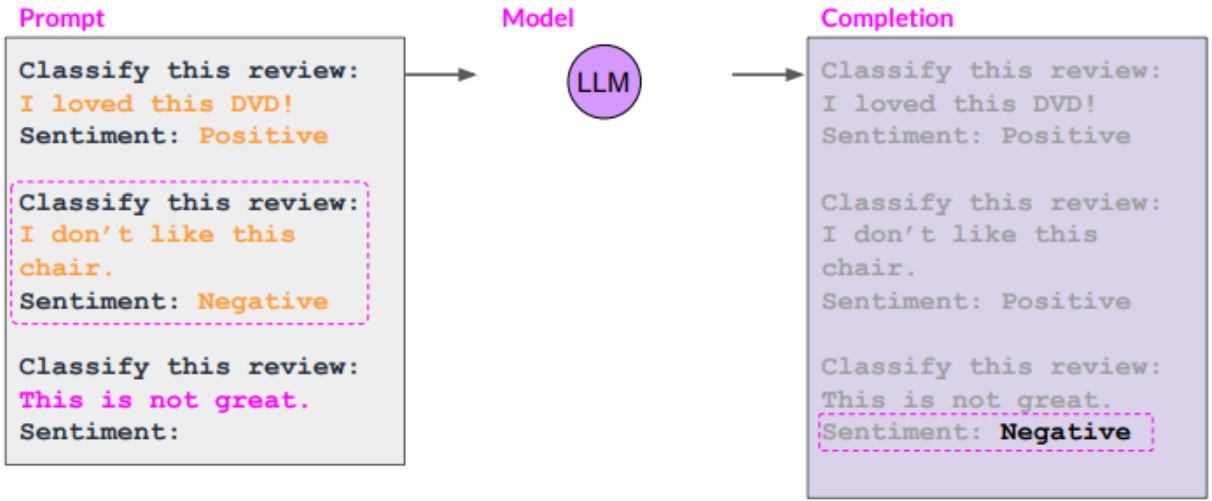
- Give the task instruction, but no examples.



LLMs: In-Context Learning

Few-shot Prompting

- Provide a few examples before the new input.



Why Pretraining and Transfer Learning?

Training a deep model from scratch for every NLP task can be challenging:

- Deep learning models can require substantial training data
- Labelled data may be limited in the target domain
- Source and target domains may differ
- Training large models from scratch is computationally expensive

Solution: Pretrain once, then Reuse and adapt

Transfer Learning

Use a pretrained model as a starting point for a new task or domain.

The pretrained model has already learned:

- useful representations
- general patterns
- knowledge from a large dataset

We then adapt it using a smaller task-specific dataset.

- The pretrained parameters may be frozen or updated, depending on the adaptation strategy

Pretrained Model + Task-specific Data = Adapted Model

Ex: Image classification: Knowledge gained while learning to recognize cars could be applied when trying to recognize trucks

Transfer Learning – Transductive v/s Inductive

Transductive transfer

- Target-domain **unlabelled data is available**, but target labels are not
- Use the target data to adapt the model to the target distribution
- Example**:** domain adaptation

Inductive transfer

- **Labelled data is available** for the target task
- Use the pretrained/source knowledge together with target-task labels to improve performance
- Example: pretraining, multi-task learning

Transfer Learning Strategies

Strategy	What is tuned
Feature extraction	New task-specific layers only
PEFT	Small number of added/selected parameters
Partial fine-tuning	Selected pretrained layers
Full fine-tuning	Most/all pretrained parameters
Knowledge distillation	*A separate student model learns from the teacher*

References

1. CH-7 - Speech and Language Processing by Daniel Jurafsky
2. https://www.youtube.com/watch?v=BtmsIy0j_dY&t=5s

BITS Pilani
Pilani Campus

Backup

DL Model Types

Deep Learning Model Types

Discriminative

- Used to classify or predict
- Typically trained on a dataset of labeled data
- Learns the relationship between the features of the data points and the labels

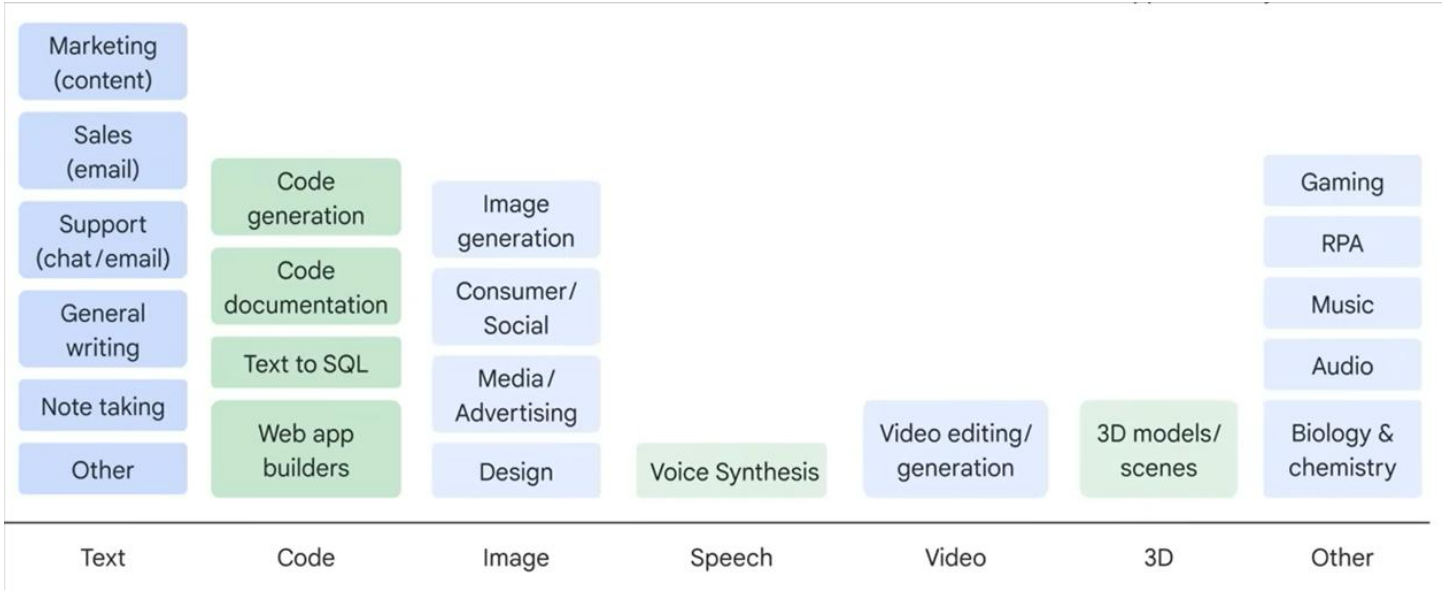
Generative

- Generates new data that is similar to data it was trained on
- Understands distribution of data and how likely a given example is
- Predict next word in a sequence

Generative AI

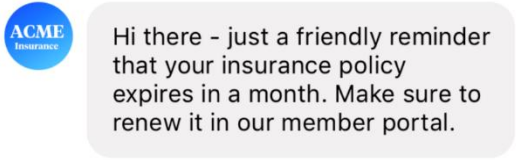
- GenAI is a type of Artificial Intelligence that creates new content based on what it has learned from existing content.
- The process of learning from existing content is called training and results in the creation of a statistical model.
- When given a prompt, GenAI uses this statistical model to predict what an expected response might be–and this generates new content.

Generative AI Applications

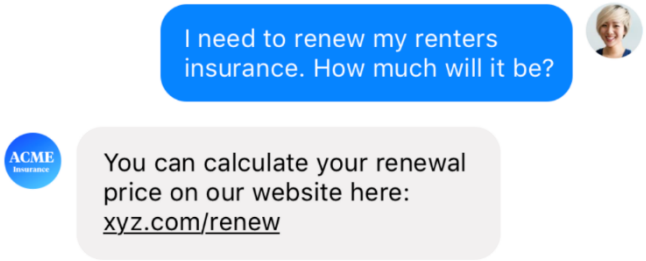


Generative AI- AI Assistants

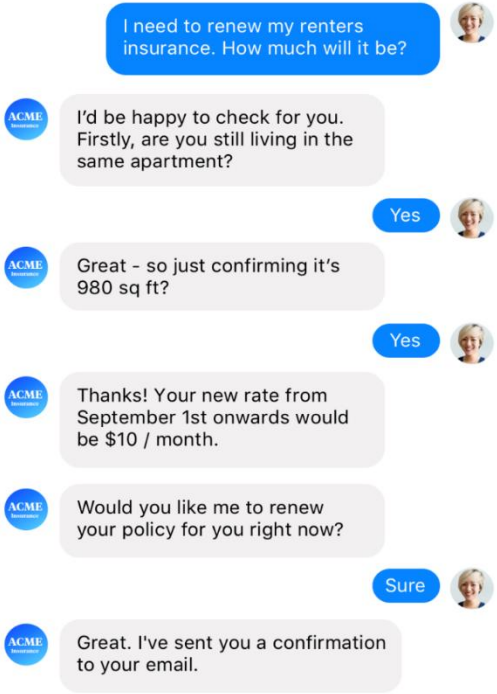
Notification Assistant


ACME Insurance
Hi there - just a friendly reminder that your insurance policy expires in a month. Make sure to renew it in our member portal.

FAQ Assistant


I need to renew my renters insurance. How much will it be?
ACME Insurance
You can calculate your renewal price on our website here: xyz.com/renew

Contextual Assistant

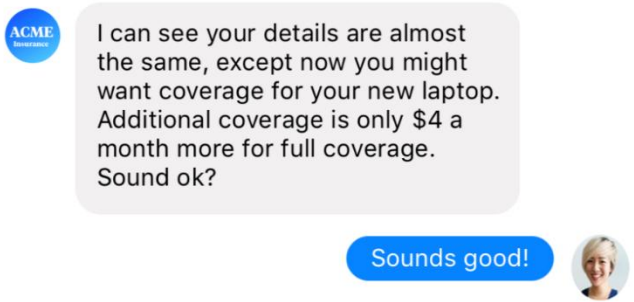

I need to renew my renters insurance. How much will it be?
ACME
I'd be happy to check for you. Firstly, are you still living in the same apartment?
Yes
Great - so just confirming it's 980 sq ft?
Yes
Thanks! Your new rate from September 1st onwards would be $10 / month.
Would you like me to renew your policy for you right now?
Sure
Great. I've sent you a confirmation to your email.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Generative AI- AI Assistants

Personalized Assistant

- Assistant knows you much more in detail
- Quickly checks a few final things before giving you a quote tailored to your actual situation.


ACME
I can see your details are almost the same, except now you might want coverage for your new laptop. Additional coverage is only $4 a month more for full coverage. Sound ok?
Sounds good!

Autonomous Organization of Assistants

- Group of AI assistants that know every customer personally
- Eventually run large parts of company operations—from lead generation over marketing, sales, HR, or finance

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

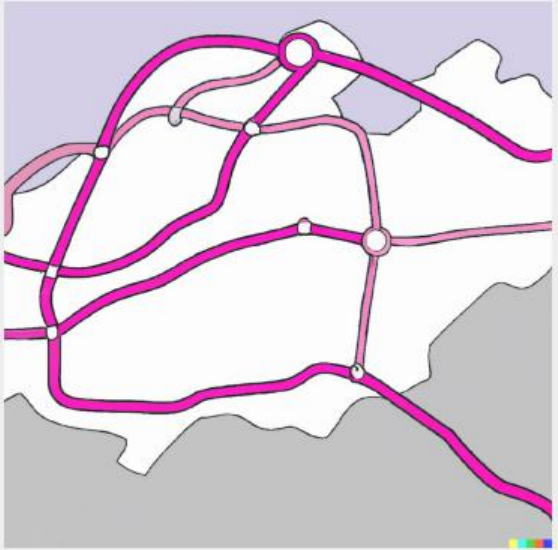
Generative AI- PaintBox

What do you want to create?

An imaginary subway map in a coastal city.

Image dimensions: by (Max 2048)

Generate



Generative AI- CodeAid

```
1  def binary_search(arr, x, l, r):
2      if r >= l:
3          mid = l + (r - l) // 2
4          if arr[mid] == x:
5              return mid
6          elif arr[mid] > x:
7              return binary_search(arr, x, l, mid - 1)
8          else:
9              return binary_search(arr, x, mid + 1, r)
       else:
           return -1
```

< 1/2 > Accept Tab

Why Computation Graphs

- For training, we need the derivative of the loss with respect to each weight in every layer of the network
- But the loss is computed only at the very end of the network!
- Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)
- **Backprop** is a special case of **backward differentiation**
- Which relies on **computation graphs**.

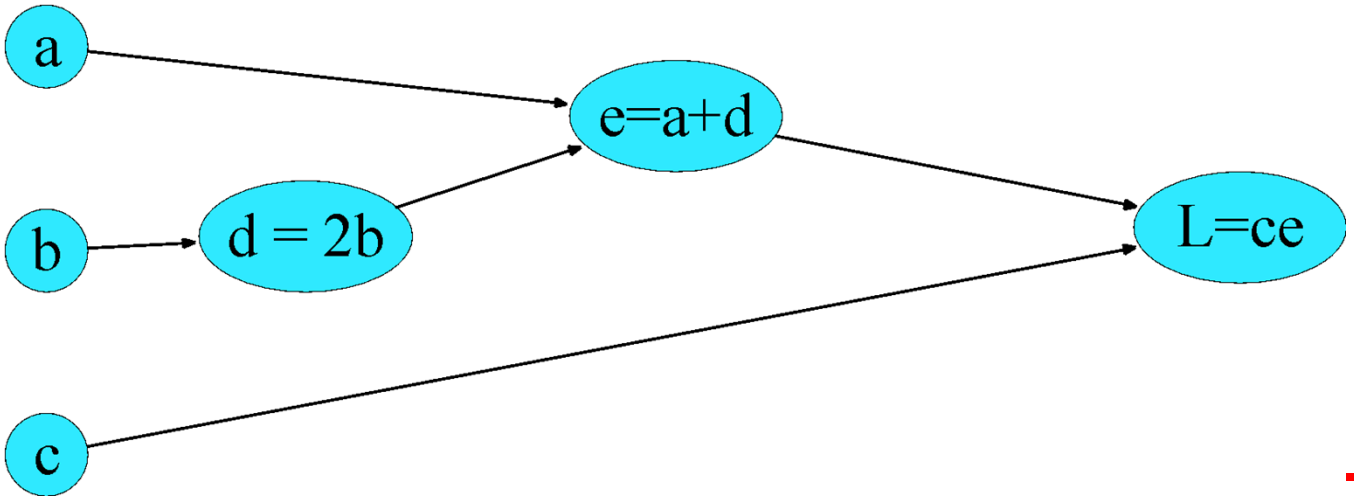
Computation Graphs

- A computation graph represents the process of computing a mathematical expression

$$L(a,b,c) = c(a+2b)$$

Computations:

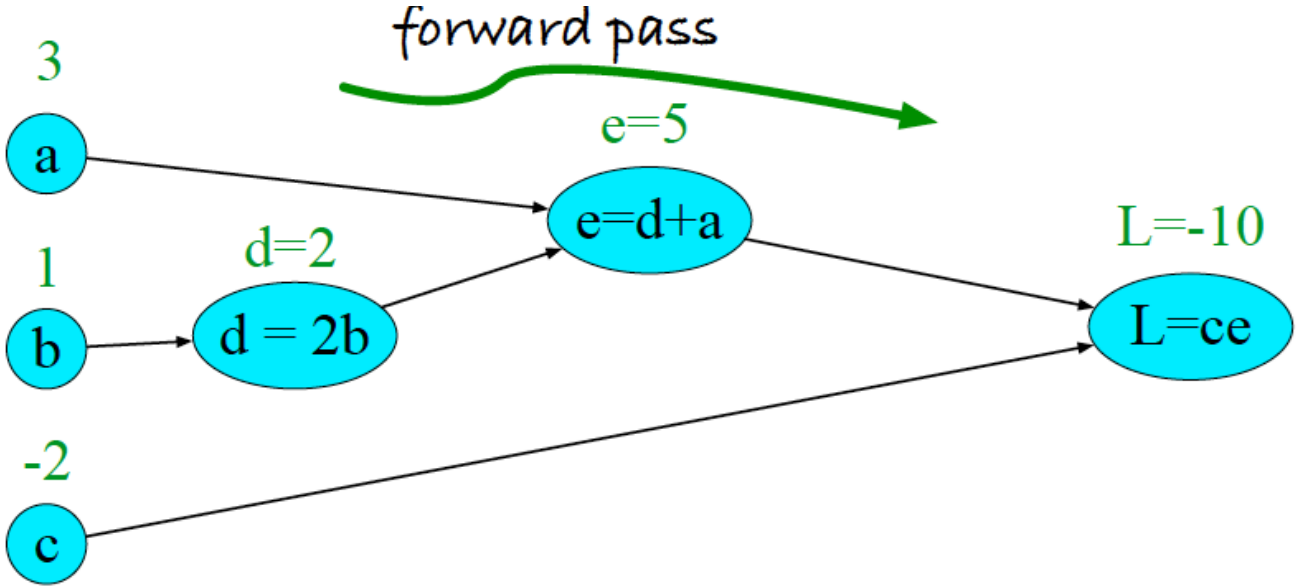
$$d = 2 * b$$
$$e = a + d$$
$$L = c * e$$



$$L(a,b,c) = c(a+2b)$$

Computations:

$$d = 2 * b$$
$$e = a + d$$
$$L = c * e$$



Backwards differentiation in computation graphs

- The importance of the computation graph comes from the backward pass
- This is used to compute the derivatives that we'll need for the weight update.

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

We want: $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$

The derivative $\frac{\partial L}{\partial a}$, tells us how much a small change in *a* affects *L*.

The chain rule

- Computing the derivative of a composite function:

- $f(x) = u(v(x))$ $\quad \dfrac{df}{dx} = \dfrac{du}{dv} \cdot \dfrac{dv}{dx}$

- $f(x) = u(v(w(x)))$ $\quad \dfrac{df}{dx} = \dfrac{du}{dv} \cdot \dfrac{dv}{dw} \cdot \dfrac{dw}{dx}$
-

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

$$\frac{\partial L}{\partial c} = \epsilon$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

Example

$$L(a,b,c) = c(a+2b)$$

$$d = 2*b$$

$$e = a+d$$

$$L = c*e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a+d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$

Example

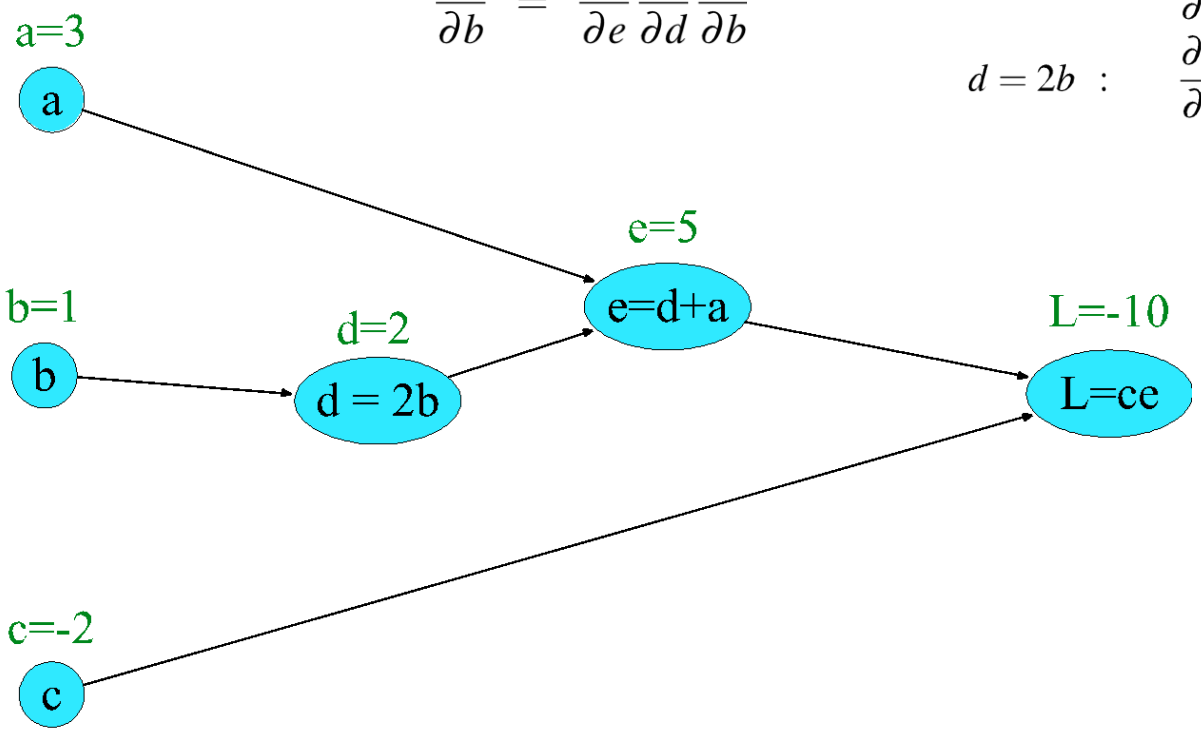
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

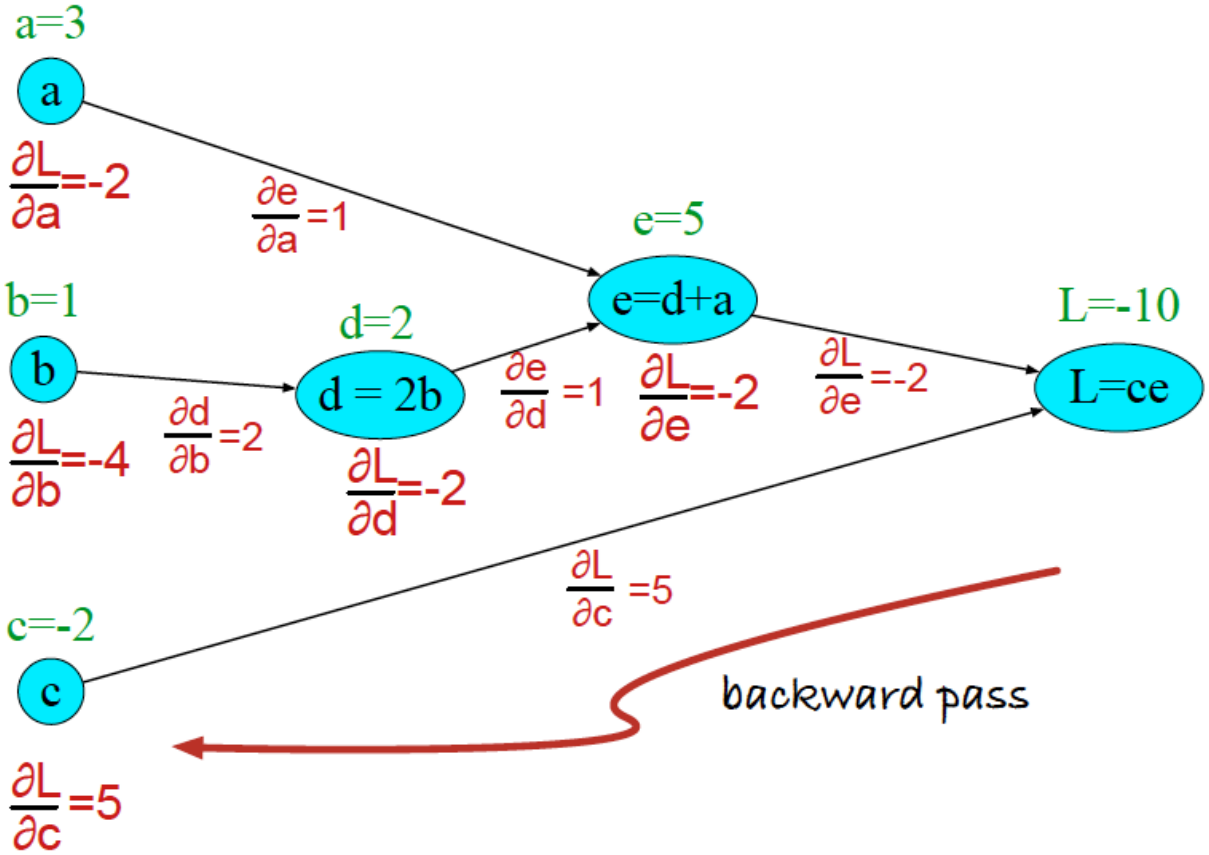
$$L = ce \;:\; \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a + d \;:\; \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

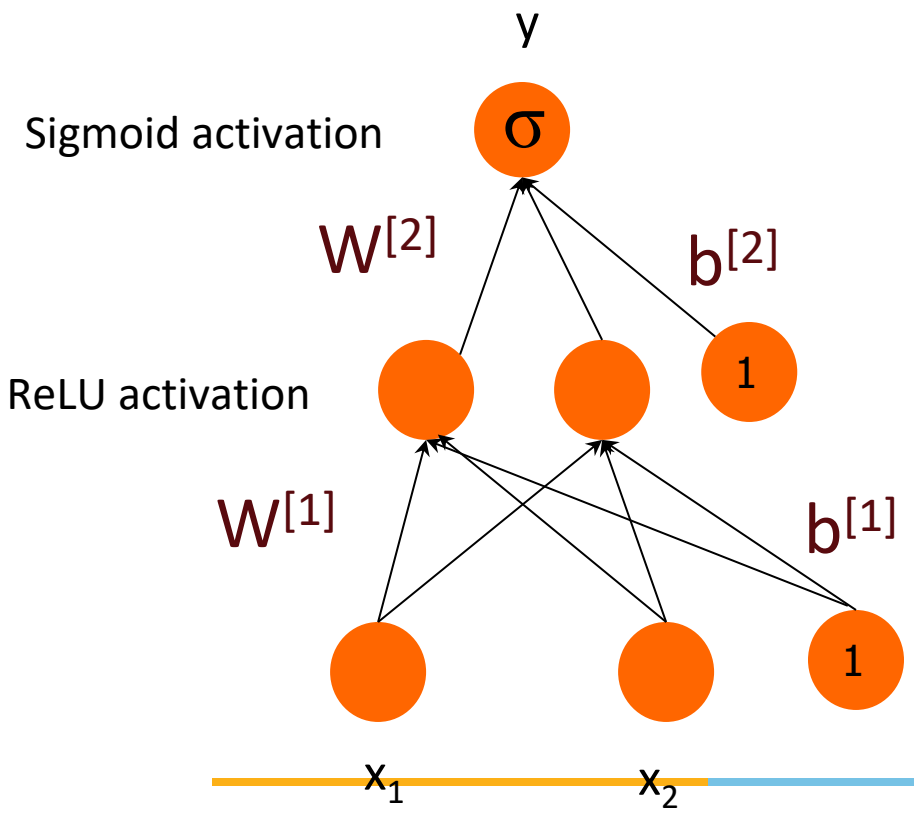
$$d = 2b \;:\; \frac{\partial d}{\partial b} = 2$$



Example


a=3
a
∂L/∂a=-2
∂e/∂a=1
e=5
e=d+a
b=1
b
∂L/∂b=-4
∂d/∂b=2
d=2
d = 2b
∂e/∂d=1
∂L/∂e=-2
∂L/∂e=-2
L=-10
L=ce
∂L/∂d=-2
∂L/∂c=5
c=-2
c
backward pass
∂L/∂c=5

Backward differentiation on a two layer network



$$z^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}$$
$$a^{[1]} = \text{ReLU}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$
$$\hat{y} = a^{[2]}$$

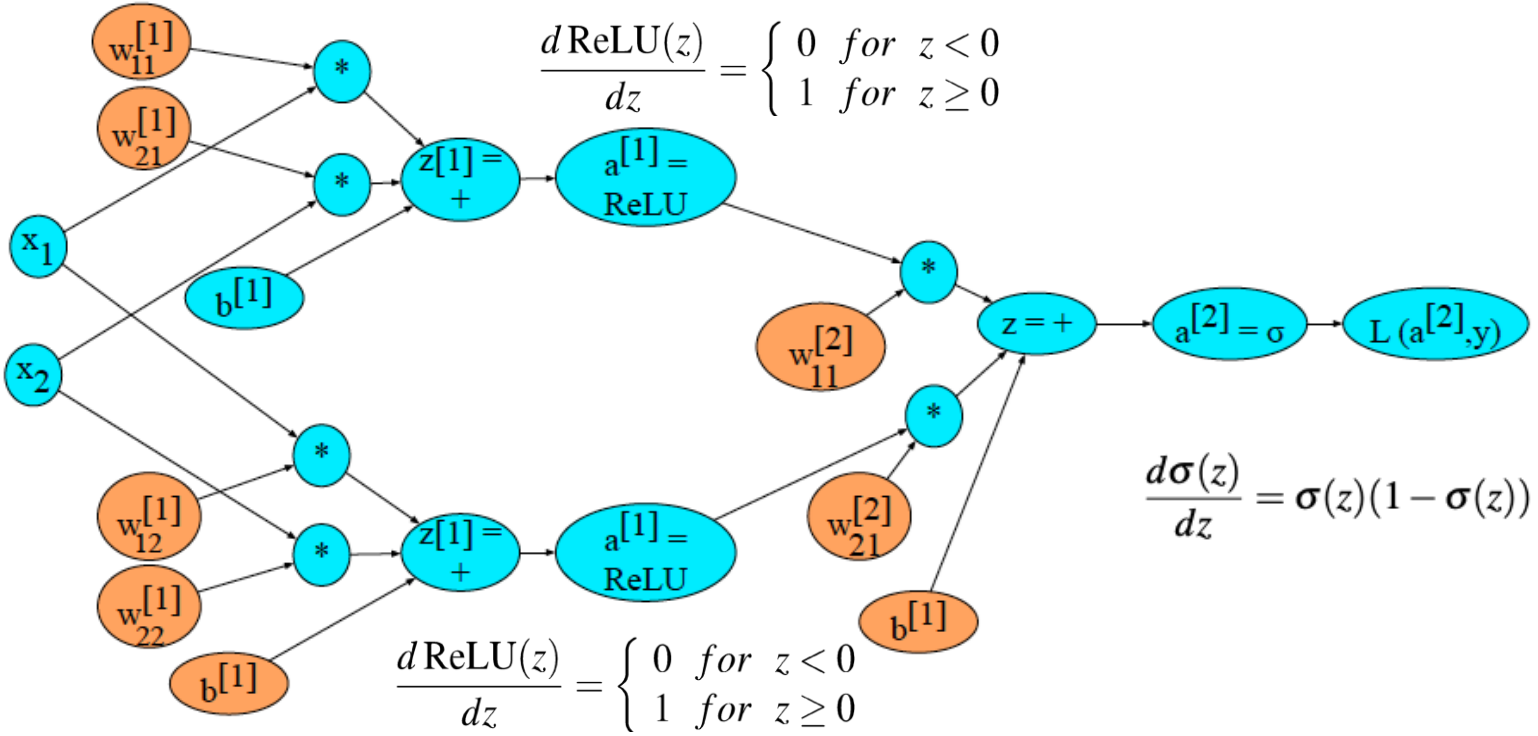
Backward differentiation on a two layer network

$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \text{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

$$
\frac{d\,\text{ReLU}(z)}{dz} = \begin{cases} 0 & for \;\; z < 0 \\ 1 & for \;\; z \geq 0 \end{cases}
$$

$$
\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))
$$

Backward differentiation on a 2-layer network



Starting off the backward pass: $\frac{\partial L}{\partial z}$

$$z^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}$$
$$a^{[1]} = \text{ReLU}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$
$$\hat{y} = a^{[2]}$$

$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

$$L(a, y) = -(y \log a + (1-y)\log(1-a))$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}$$

$$\frac{\partial L}{\partial a} = -\left(\left(y\frac{\partial \log(a)}{\partial a}\right) + (1-y)\frac{\partial \log(1-a)}{\partial a}\right)$$

$$= -\left(\left(y\frac{1}{a}\right) + (1-y)\frac{1}{1-a}(-1)\right) = -\left(\frac{y}{a} + \frac{y-1}{1-a}\right)$$

$$\frac{\partial a}{\partial z} = a(1-a) \qquad \frac{\partial L}{\partial z} = -\left(\frac{y}{a} + \frac{y-1}{1-a}\right)a(1-a) = a - y$$